University of Bergen

Discrete hidden Markov models with application to stock trading algorithms

Master of Science in Actuarial Science

Andreas Hansen

January 2021

Abstract

Attempting to integrate discrete hidden Markov models into stock trading algorithms by interpreting the S&P500 Index closing prize as an stock and then separating closing prizes into distinct categories. This was successfully done by [2] and we look to replicate these results. Additionally, we look at viability of viewing hidden states in a stock trading algorithm as signals for the correct stock market behaviour, where we adjust our position in the stock market accordingly to the predicted next state.

Acknowledgements

Firstly, I would like to thank my supervisor, Andreas Stordal, for great guidance during this thesis and always being available for questions. I would also like to thank my family for always being such an amazing support for me, especially during this last year, and my fellow student, for all the time we spent together during these years of studying.

Contents

1 Introduction **4**

2 Hidden Markov Models **6**

2.1 Markov Process 6

2.1.1 Discrete time 7

2.1.2 Continuous time 10

2.2 Hidden Markov Models 12

3 Estimating Hidden States **14**

3.1 Viterbi Algorithm 14

3.1.1 Overview 14

3.1.2 Accuracy 20

3.1.3 Extending Viterbi to forecasting 26

4 Parameter estimation **27**

4.1 EM-algorithms 27

4.2 Baum-Welch Algorithm 28

5 A Brief Introduction to the Stock Market and Financial Data **32**

5.1 Time Series 32

5.1.1 Short introduction to Time Series 32

5.1.2 Stationary time series 34

5.1.3 Financial time series 36

5.2 Stock Market 36
5.2.1 Introduction 36
5.2.2 Bull- and Bear market 38
5.2.3 Investment strategies 39

6 Application of HMMs in stock trading algorithms 41
6.1 Our Data 41
6.2 HMM architecture 43
6.2.1 Transforming Observations 43
6.2.2 Defining the Hidden States 44
6.2.3 Estimating hidden states and forecasting 45
6.2.4 Multiple HMMs 46
6.2.5 Training the HMMs 48
6.3 Decision Algorithm 50
6.4 Results 52
6.4.1 Overall results for testing period 52
6.4.2 Estimated HMMs 58

7 Conclusions and future work 61
7.0.1 Future work 63

A Working with HMMs in R 66
A.1 Generate a HMM problem 66
A.2 Forecast the next state using Viterbi and HMMs 69

Chapter 1

Introduction

The goal of this thesis is to reflect on "hidden Markov models" (HMM) viability as a component in stock trading algorithms for discrete time models. However, before we study the application, a foundation of the theoretical principles surrounding discrete time HMMs will be given.

This thesis is outlined as follows. Chapter 2 gives an overview of Markov models in both discrete- and continuous time using discrete observations, however the main focus is on discrete time models which gives a natural focus on the main topic, Hidden Markov Models.
Chapter 3 revolves around decoding hidden states given a string of observations, where we are supported by a complete HMM. This is done by using the Viterbi algorithm.
Chapter 4 provides an approach for estimating parameters of a HMM when they are unknown using the Baum-Welch algorithm.
Chapter 5 gives a brief overview of time series and stock market data, while chapter 6 merge together the theory presented in previous chapters. We look at previous attempts of including discrete time HMMs in stock trading algorithms. On the basis of these previous attempts we have, significantly inspired by [2], created our own algorithms. One of the algorithms are a close replica of the

model described in [2], while the other algorithm have some adjustments. The main goal is to study the effects of these changes. The thesis is concluded with a summary where we make our final notes and motivate for future work on the basis of our results.

Chapter 2

Hidden Markov Models

The goal of most mathematical model is to give an explanation of a phenomenon. The form of a mathematical model depends on the phenomenon at hand. In this thesis the main concerned is stochastic models. Unlike deterministic models, which predicts a single outcome from a given set of circumstances, a stochastic model predicts a set of possible outcomes weighted by their likelihoods and probabilities (M. Pinsky & S. Karlin, chapter 1, 2011)[4].

This chapter gives an overview of stochastic modeling using Markov models. Section 2.1 introduces the criteria for a stochastic model to be a Markov model and we discuss important characteristics revolving Markov models. In Section 2.2 hidden Markov models are introduced, where there is an added layer of observations generated from the hidden model. In this thesis we only consider models where the state space is finite. Remark that we are only interested in Markov with a finite state space in this thesis.

2.1 Markov Process

This section gives a brief overview Markov processes for the reader to recall the essentials. A Markov process is memoryless, which means that the process does

not depend on the past. In other words, if the process find itself in a given state, it does not matter how it got there if we are interested in forecasting the next step(s), only the fact that the process currently find itself in that given state is needed. Formally speaking, the Markov property is

$$P(X_{t+1} = j | X_0 = i_0, ..., X_{t-1} = i_{t-1}, X_t = i) = P(X_{t+1} = j | X_t = i), \\ \text{where } t = 1, ..., T \tag{2.1}$$

and X_t is the state of the model at time t, while $i, j \in S$ with S being the state space.

2.1.1 Discrete time

A discrete time Markov Process is defined on a set of discrete time indicies, denoted $t, t+1$ etc. Everything happening between time t and time $t+1$ is considered to occur at time $t+1$. As an example, let us consider the following transition matrix where we assume a finite set of possible outcomes

$$\begin{array}{c|ccc|} & 1 & 2 & 3 \\ 1 & \mathrm{P}_{11} & \mathrm{P}_{12} & \mathrm{P}_{13} \\ 2 & \mathrm{P}_{21} & \mathrm{P}_{22} & \mathrm{P}_{23} \\ 3 & \mathrm{P}_{31} & \mathrm{P}_{32} & \mathrm{P}_{33} \\ \end{array},$$

where P_{ij} is the probability of the process travelling to state j at time $t+1$ given that the process is in state i at time t. Intuitively, the following equality has to hold due to the law of total probability

$$\sum_{j=1}^{N} P_{ij} = 1, \quad j = 1, .., N. \tag{2.2}$$

Classification of States

A discrete time Markov Process, with finite state space, is said to be *irreducible* if the all states communicate with each other, meaning the process can not be separated as illustrated below

	1	2	3	4
1	1	0	0	0
2	0.1	0.9	0	0
3	0	0	0.5	0.5
4	0	0	0.4	0.6

The matrix above could be separated into two matrices and hence the matrix is not irreducible. The *period*, denoted as d, of each state is another useful definition. The period is defined as the shortest possible time for the process to return to a given state. Mathematical speaking, a state have a period $d = 1$ if the following fulfilled

$$P_{ii} > 0. \tag{2.3}$$

Futhermore, if every state in a given transition matrix have a period of $d = 1$, the matrix is said to be *aperiodic*. The last classification introduced in this section is *recurrency*. A state is said to be recurrent if $f_{ii} = 1$, where f_{ii} is the probability of eventually (given an unlimited time horizon) returning to state i given that the process currently is in state i.

The Stationary Distribution

A discrete Markov Process is generally defined at time $t = 1, .., T$, where $T \leq \infty$. We enter the process at a random time k, where $1 \leq k \leq T$. The probability of

enter the process at any given state is determined by the *stationary distribution* of the process. The stationary distribution is defined as the marginal probability distribution $P(X_t = j)$ independent of t. From here on we will refer to the stationary distribution as π. The stationary distribution for an N state Markov process is given as the vector $\pi = (\pi_1, .., \pi_N)$. To obtain π, we have to solve the two following equations

$$\pi_i = \sum_{j=1}^{N} \pi_j P_{ij}, \qquad i = 1, ..N, \tag{2.4}$$

$$\sum_{j=1}^{N} \pi_j = 1. \tag{2.5}$$

Equation (2.4) creates an expression for each π_i expressed by the other π_is. Then the equations is solved by substituting terms and the incorporation of equation (2.5). As an example, let us consider the following 3x3 transition matrix

	1	2	3
1	0.7	0.2	0.1
2	0.1	0.6	0.3
3	0.2	0.4	0.4

From equation (2.4) we get

$$\begin{aligned} \pi_1 &= 0.7\pi_1 + 0.1\pi_2 + 0.2\pi_3 \\ 0.3\pi_1 &= 0.1\pi_2 + 0.2\pi_3 \\ \pi_1 &= \frac{1}{3}\pi_2 + \frac{2}{3}\pi_3 \end{aligned}$$

$$
\begin{aligned}
\pi_2 &= 0.2\pi_1 + 0.6\pi_2 + 0.4\pi_3 \\
0.4\pi_2 &= 0.2\pi_1 + 0.4\pi_3 \\
0.4\pi_2 &= 0.2(\frac{1}{3}\pi_2 + \frac{2}{3}\pi_3) + 0.4\pi_3 \\
\frac{1}{3}\pi_2 &= \frac{8}{15}\pi_3 \\
\pi_2 &= \frac{8}{5}\pi_3
\end{aligned}
$$

$$
\begin{aligned}
\pi_3 &= 0.1\pi_1 + 0.3\pi_2 + 0.4\pi_3 \\
0.6\pi_3 &= 0.1\pi_1 + 0.3\frac{8}{5}\pi_3 \\
\frac{3}{25}\pi_3 &= 0.1\pi_1 \\
\frac{6}{5}\pi_3 &= \pi_1
\end{aligned}
$$

Thus, we have $\pi = (\frac{6}{5}\pi_3, \frac{8}{5}\pi_3, \pi_3)$. Next we use equation (2.5). $(\frac{6}{5} + \frac{8}{5} + 1)\pi_3 = 1, \quad \pi_3 = 5/19$, and hence, $\pi = (\frac{6}{19}, \frac{8}{19}, \frac{5}{19})$ is the stationary distribution of this Markov process.

The stationary distribution can be considered the initial/starting distribution, given the assumption that the process is constantly running. For instance, let us assume the weather in a city can be described as an discrete Markov process, which is measured in either rainy days or not rainy days. We arrive to this city without information of the weather the previous day. This leads to us arriving at any random day of this discrete Markov process and thus the probability of arriving at a rainy day is equivalent to the long term proportions of rainy days.

2.1.2 Continuous time

Here, we briefly discuss continuous time Markov models. Fundamentally, the difference compared to the discrete Markov model is the state transition times. Unlike the discrete time Markov model, which only transition at distinct time points t, where $t = 0, 1, 2, ..., T$, the continuous Markov model can one from one state to another for any $t > 0$, where t can be any real number on the interval

from 0 to T. Let $S = (1, ...N)$ be the finite state space. The transition probability $P_{ij}(t)$ for a continuous Markov process satisfies

$$\begin{aligned}
&\textbf{(a)} P_{ij} \geq 0, \\
&\textbf{(b)} \sum_{j=0}^{N} P_{ij}(t) = 1, \quad i, j \in S, \\
&\textbf{(c)} P_{ik}(s+t) = \sum_{j=0}^{N} P_{ij}(s) \cdot P_{jk}(t) \quad \text{for } t, s \geq 0, \\
&\textbf{(d)} \lim_{t \to 0+} P_{ij} = 1, i = j, \quad \lim_{t \to 0+} P_{ij} = 0, i \neq j.
\end{aligned} \tag{2.6}$$

Equation (2.6 (c)) is known as the *Chapman-Kolmogorov relation*[4]. In continuous time Markov models, the transition probabilities can no longer be defined in a static manner given the possibility of transitioning at any time, thus two new definitions are introduced

$$\begin{aligned}
&\lim_{t \to 0+} \tfrac{1 - P_{ii}(t)}{t} = q_i, \\
&\lim_{t \to 0+} \tfrac{P_{ij}(t)}{t} = q_{ij}.
\end{aligned} \tag{2.7}$$

The interpretation of q_i is the rate of which the process leaves state i and q_{ij} is the rate of which the process transitions from state i to state j. Now, let $\Delta t \to 0$ and X_t denote the state at time t, then we obtain the following equations

$$P(X_{t+\Delta t} = i | X_t = i) = 1 - q_i \cdot \Delta t + o(h), \tag{2.8}$$

$$\begin{aligned}
&P(X_{t+\Delta t} = j | X_t = i) = q_{ij} \cdot \Delta t + o(h), \\
&\text{for } i \neq j.
\end{aligned} \tag{2.9}$$

Equation (2.8) and Equation (2.9) is famously known as the *infinitesimal description* of the continuous Markov process. The *infinitesimal matrix* is an commonly a preferred way of describing the state transition process when dealing with continuous Markov models. Given Equation (2.6 (a)), the process will inevitably leave state i. Thus, the we define $p_{ij} = q_{ij}/q_i$, $i \neq j$ as the probability of *jumping* from state i to state j. The infinitesimal matrix for a three

state continuous Markov model is defined as follows

$$\mathbf{A} = \begin{Vmatrix} -q_1 & q_{12} & q_{13} \\ q_{21} & -q_2 & q_{23} \\ q_{31} & q_{32} & -q_3 \end{Vmatrix}.$$

In Section 2.1.1, the stationary distribution of a discrete time Markov model was discussed. For a continuous Markov model, stationary distribution is obtained by solving $\pi\mathbf{A} = 0$.

2.2 Hidden Markov Models

A Hidden Markov Model (HMM) is a Markov model that have latent, unobserved states and is the underlying driver of the visible observation process. In other words, in conjunction with the now hidden *State transition process* there is also an *Observation emission process*. In this section we will introduce the framework of a HMM.

In addition to the characteristics of a Markov Model, which were explained in the previous sections, we now have the added layer of observations generated by the states. An observation emission process with the following observation space $\mathbf{Y} = (A, B, C)$, generated by hidden states following state space $\mathbf{S} = (s_1, s_2, s_3)$ can be described in a matrix format in the following way.

	A	B	C
s_1	$P[Y = A\|S = 1]$	$P[Y = B\|S = 1]$	$P[Y = C\|S = 1]$
s_2	$P[Y = A\|S = 2]$	$P[Y = B\|S = 2]$	$P[Y = C\|S = 2]$
s_3	$P[Y = A\|S = 3]$	$P[Y = B\|S = 3]$	$P[Y = C\|S = 3]$

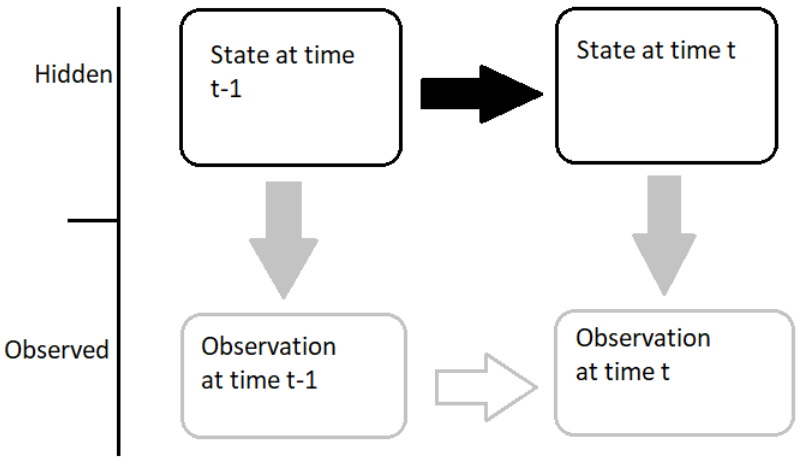


Figure 2.1: Illustration of a HMM behaviour over two time periods

The state transition matrix have the same characteristics that were presented in section 2.1, but now we are not able to observe it. The observation emission matrix is the connecting piece between the observed sequence and the hidden state process.

In Chapter 3, algorithms for determining the hidden states are discussed. Often, the parameters of the HMM are unknown and have to be estimated. This is discussed in detail in Chapter 4. Many real world situations could be interpreted as a HMM environment. In Chapter 6 we will make case for interpreting stock prices as a HMM environment, where we assume there is a hidden state process generating the changes in stock prizes.

Chapter 3

Estimating Hidden States

The previous chapter introduced the framework of a HMM where we have the state transition matrix and the corresponding observation emission matrix. This chapter describes an algorithm for estimating the hidden states at each time point using the information given by the model in conjunction with a sequence of observations, famously known as the Viterbi algorithm.

3.1 Viterbi Algorithm

The Viterbi Algorithm was first introduces in (A. Viterbi, 1967)[5] and has become popular in many fields of application since. This section gives a theoretical overview of the algorithm followed by an application example. Furthermore, the accuracy of the algorithm is evaluated on several different HMMs.

3.1.1 Overview

The Viterbi Algorithm is a tool for estimating the state of a HMM by finding the most probable path given the observations at hand, the state matrix and the observation matrix. The stationary distribution is a requirement as well. The probability of each sequence of states can be computed using the Markov properties and the conditional probabilities of the observations, however, this

grows exponentially with time and space. The Viterbi Algorithm solves this by using dynamic programming. In general, the observation sequence can get very large, hence the probabilities of the paths converges to zero. The Viterbi algorithm circumvents this by log-transform to the probabilities. The path with the highest log-transformed value is the solution of the Viterbi algorithm, since the logarithm is a monotone transformation and thus have the same optimum.

Algorithm 1: Viterbi Algorithm

Initial values: State space $S = (s_1, s_2, ..., s_K)$, observation space $O = (o_1, o_2, ..., o_N)$, state transition matrix A where a_{kj} is the transition probability from state i to state j, observation emission matrix B where $b_i(o_n)$ is the probability of seeing o_n in state i, initial state distribution π_0 and the observed sequence Y. v is a matrix storing the most likely path in state j log-transform probability at each time point, $p_j(y_i|t = i)$ denotes the probability of observing y_i at time $t = i$ in state j.

for $j \leq K$ **do**
  $p_j(y_1|t = 1) < -\log(\pi_0[j] \cdot b_j(y_1))$
  v[j,1] <- $\mathrm{p}_j(y_1|t = 1)$
end
while $2 \leq i \leq N$ **do**
  for $j \leq K$ **do**
    $p_j(y_i|t = i) < -\log(b_j(y_i)) \cdot \max_{k=1,..,K}(v[k, t-1] + \log(a_{kj}))$
    backpointer[i-1,j] <- $\arg\max_k(v[k, t-1] \cdot a_{kj})$
    v[j,i] <- $p_j(y_i|t = i)$
  end
end
Backtrack the best path for the most probable state at $t = N$ using the backpointer all the way to $t = 1$.
Result: Most probable path $X = (x_1, x_2, ..., x_N)$

The algorithm starts by evaluating the first observation and calculate the likelihood of being in each state $s_j, j = 1, ..., K$, given by (3.1). These calculations are saved in a matrix format v for calculating the remain stages of the paths

$$p_j(y_1|t = 1) = p_j(y_1) \cdot \pi_0[j]. \tag{3.1}$$

From this point, the goal is to find the optimal path. To obtain the optimal path we have to calculate all paths that might lead to the most likely path at $t = N$, ignoring paths without the potential of being optimal. This is done as following

$$p_j(y_i|t = i) = p_j(y_i) \cdot \max_{k=1,...,K}(v[k, t-1] \cdot A_{kj}). \tag{3.2}$$

The difference between (3.1) and (3.2) is the second factor. When $t = i, i = 2, ..., N$, the algorithm finds a best previous state for each s_j, which is determined by which state at time $t = i-1$ maximizes the probability of being in s_j at time $t = i$. To obtain which state at $t = i-1$ is the best for s_j, the algorithm calculates the product of $v[k|i-1]$ and A_{kj}, $v[k|i-1]$ is the probability of the most likely path ending in s_k at $t = i-1$ and A_{kj} is the transition probability from s_k to s_j given by the transition matrix A. The best previous estimate for each state is stored in another matrix (backpointer in Algorithm 1) and will be used later for backtracking to find the most likely path.

Once Viterbi is done calculating for all t, we inevitably end up with a most likely state at $t = N$ for our paths. The state corresponding to the highest value is chosen as x_N.

$$x_N = \arg \max_{j=1,...,K}(p_j(y_N|t = N)). \tag{3.3}$$

It is important to take notice of what this value actually describes, which is the

probability of the candidate path which ends up in x_N. This is not the overall probability of being in that given state at $t = N$.

The algorithm moves backwards in time, starting with the solution for x_N we find the solution for $x_i, i = N-1, N-2, ..., 1$. Starting from x_{N-1} and working our way backwards, we use the backpointer to assign the x_i's. We have now obtained the most likely path.

$$x_{i-1} = backpointer[i, \arg\max x_i], i = 2, .., N \tag{3.4}$$

A simple example

Let us consider case with three hidden states, $S = (Z_1, Z_2, Z_3)$, and three possible observations, $O = (A, B, C)$. The transition matrix, T, is given by

$$T = \begin{array}{c|ccc} & Z_1 & Z_2 & Z_3 \\ \hline Z_1 & 0.5 & 0.3 & 0.2 \\ Z_2 & 0.3 & 0.6 & 0.1 \\ Z_3 & 0.2 & 0.2 & 0.6 \end{array},$$

and the corresponding emission matrix, E, given by

$$E = \begin{array}{c|ccc} & A & B & C \\ \hline Z_1 & 0.7 & 0.1 & 0.2 \\ Z_2 & 0.2 & 0.6 & 0.2 \\ Z_3 & 0.1 & 0.1 & 0.8 \end{array}.$$

The task is to find the optimized path given an observation sequence $Y = (B, B, A, C, A, A, C)$, were one observation occur for each $t = 1, .., 7$. Following

the Viterbi algorithm we start by estimating the most likely state at $t = 1$. Next, the stationary distribution is calculated $\pi_0 = (0.342, 0.391, 0.269)$. Now we have to solve (3.1), however with the Viterbi algorithm recall that in most real life scenarios log-transform the probabilities to avoid computational error when the observation sequence become long and each path might converge to zero. We solve the log-transformed equation (3.5) for every state in S.

$$\log(p_j(y_1|t = 1)) = \log(p_j(y_1)) + \log(\pi_0[j]) \tag{3.5}$$

We obtain $p_{Z_1}(B|1) = -3.3745$, $p_{Z_2}(B|1) = -1.4492$ and $p_{Z_3}(B|1) = -3.6157$. Likewise we log-transform (3.2) which result in equation (3.6)

$$\log(p_j(y_i|t = i)) = \log p_j(y_i) + \log(\max_{k=1,...,K}(v[k, t-1] \cdot T_{kj})) \tag{3.6}$$

When $t = 2$ we find the best previous state for each of the three states determined by which previous state that maximizes $v[k, t-1] \cdot T_{kj}$, which turns out to be Z_2 for all states and thereby Z_2 is stored in the backpointer matrix for each state. The complete Viterbi calculations are $p_{Z_1}(B|2) = -4.9558$, $p_{Z_2}(B|2) = -2.4709$ and $p_{Z_3}(B|2) = -6.0544$. We repeat this process for every $t = 1, ..., 7$, ending up with $p_{Z_1}(C|7) = -10.7363$, $p_{Z_2}(C|7) = -11.2471$ and $p_{Z_3}(C|7) = -10.2663$. Hence, the most likely path is in state Z_3 at $t = 7$.
Now we use the backpointer matrix to conclude the most probable path. The previous state that maximized $p_{Z_3}(C|7)$ is Z_1 and the previous state that maximized $p_{Z_1}(A|6)$ is Z_1. Tracking the optimal previous state all the way back to $t = 1$ gives us the most likely hidden state sequence

$$X = (Z_2, Z_2, Z_1, Z_1, Z_1, Z_1, Z_3)$$

The table below gives an illustration of the backtracking process. The coloured cells represent the current states best previous states.

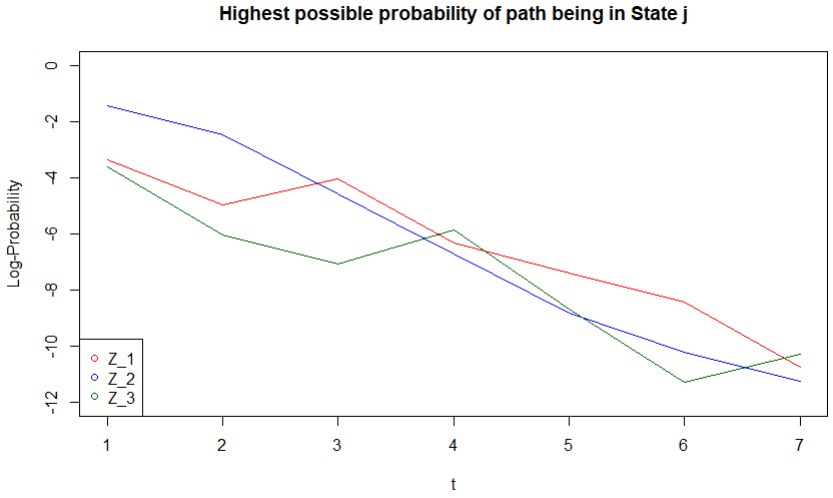


Figure 3.1

Backtracker Matrix						
Best previous state	$t = 2$	$t = 3$	$t = 4$	$t = 5$	$t = 6$	$t = 7$
For Z_1	Z_2	Z_2	Z_1	Z_1	Z_1	Z_1
For Z_2	Z_2	Z_2	Z_2	Z_2	Z_1	Z_1
For Z_3	Z_2	Z_2	Z_1	Z_3	Z_1	Z_1

It is note to remark that the most probable path do not correspond exactly with the most probable states for every t (presented in Figure 3.1) and why that is the case. In Figure 3.1 we see that state Z_3 represent the most probable state sequence at $t = 4$ however in our solution we are in state Z_1 when $t = 1$. The reason being is simply that the most probable path at $t = 4$ might not lead to the most probable path at $t = 7$, hence why we use the backpointer matrix to store information.

3.1.2 Accuracy

The previous section described how the Viterbi algorithm finds the most probable path. A question yet to be answered, however, is how well this approach actually performs. We start by taking a look at the state transition matrix T and the emission matrix E from the previous example. The observation sequences length increased to 50 observations. We use R to create a state sequence and an observation sequence and use this state sequence (see Appendix) to evaluate the performance of the Viterbi Algorithm.

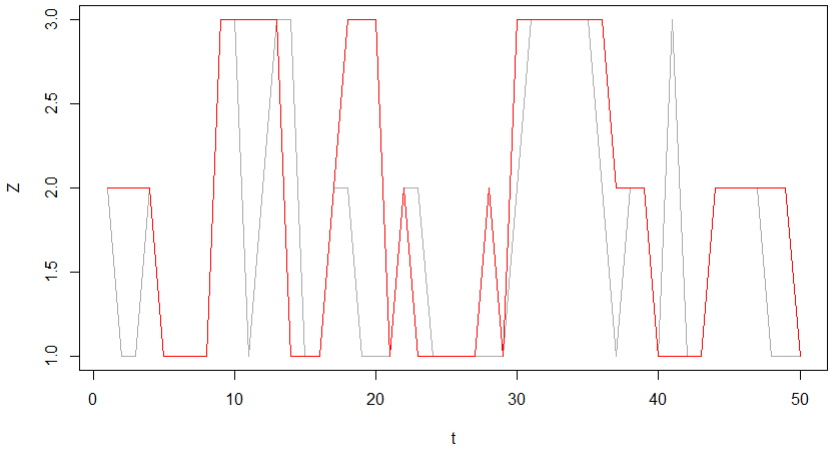


Figure 3.2: Grey line represents the true hidden state sequence generated in R (see Chapter A.1), while the red line is the most likely state sequence obtained by the Viterbi algorithm

The Viterbi solution illustrated in Figure 3.2 had an accurate classification rate of 68%. The miss-classifications are represented were the red line deviates from the grey line. However this particular hidden state sequence is not necessarily representative for the average performance of the Viterbi Algorithm, so we create 1000 different random state sequences, still using the same T and E as before with an observation sequence length $t = 1, ..., 50$, illustrated in Figure 3.3.

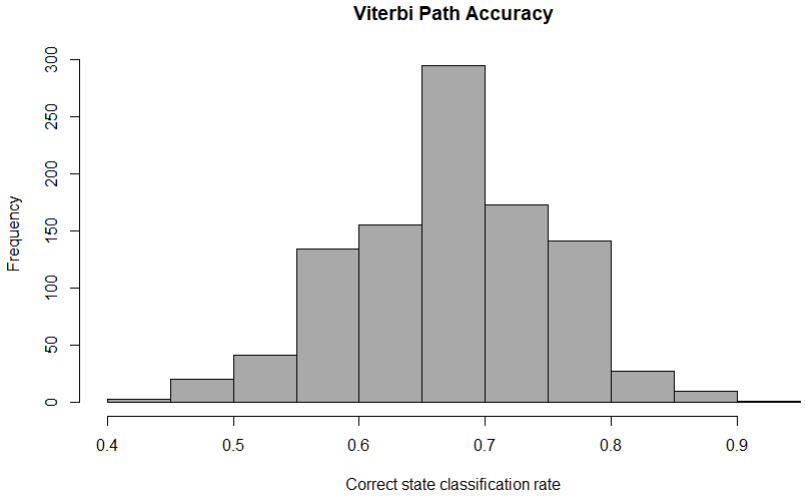


Figure 3.3: The rate of which the Viterbi path corresponds with the true hidden state

We conclude from the simulation that the Viterbi solution predicts the correct hidden state for an arbitrary t on average $67,628\%$ for this given HMM. The worst correct classification rate by the Viterbi algorithm observed in this simulation was 42%. The best performance on the other hand achieved a 92% correct classification of the hidden states. Keep in mind that this is only for hidden state sequences and observation sequences generated for T and E, with all observation sequences having a length of 50. Different HMMs will lead to different levels of success with the Viterbi algorithm. The degree of which an emission matrix correlates the observations to the hidden states inevitably dictates our classification success, which also is true for the number of expected state transfers (dictated by the state transition matrix). Furthermore, we have to pay attention to the number of states and possible observations. The bottom line is that the success of the Viterbi Algorithm is largely dependent by level of complexity in the relevant HMM. The following plots at the end of this section shows how some different HMMs and different observation sequence lengths (n)

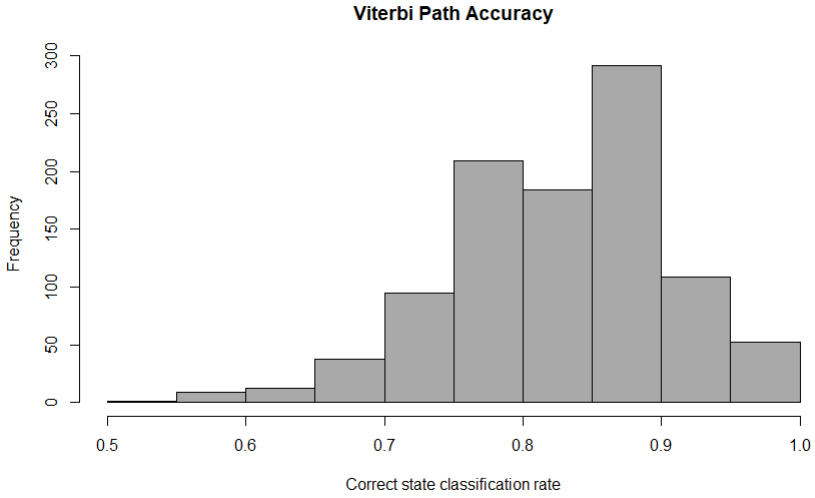


Figure 3.4: For a 3x3 state transition matrix T and a 3x3 emission matrix E with $n = 50$, where both T and E have a diagonal of 0.8 and the other element of the matrix equals 0.1. Mean success of $83,288\%$

influence the success of the Viterbi Algorithm.

In this small scale study of the Viterbi performance there are a few key take-aways. Firstly, we would much rather prefer the state transition matrix to have a random behaviour pattern than the emission matrix. If we compare Figure 3.6 and Figure 3.7 this is evident. Comparing these two results with Figure 3.4, the case with the less polarized state transition matrix only experience a slight drop in success, however the case with the less polarized emission matrix hardly preforms better than we would expect a pure random guess strategy. Secondly, the length of the observation sequence does not seem to impact the mean success

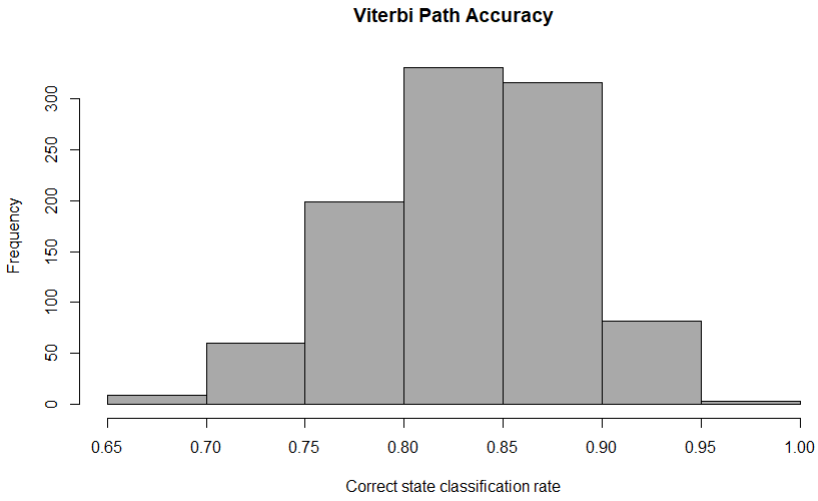


Figure 3.5: For a 3x3 state transition matrix T and a 3x3 emission matrix E with $n = 100$, where both T and E have a diagonal of 0.8 and the other element of the matrix equals 0.1. Mean success of $83,672\%$

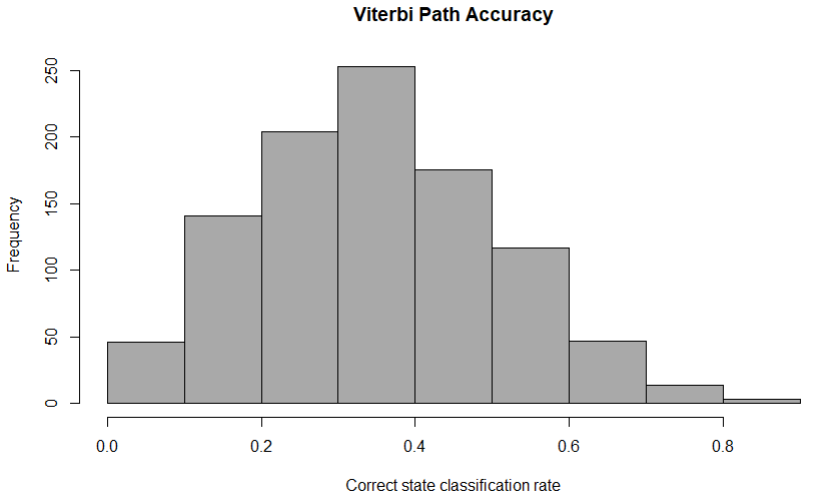


Figure 3.6: For a 3x3 state transition matrix T and a 3x3 emission matrix E with $n = 50$, where T have a diagonal of 0.8 and the other element of the matrix equals 0.1 while E have a diagonal of 0.4 and the other elements equals 0.3. Mean success of $36,068\%$

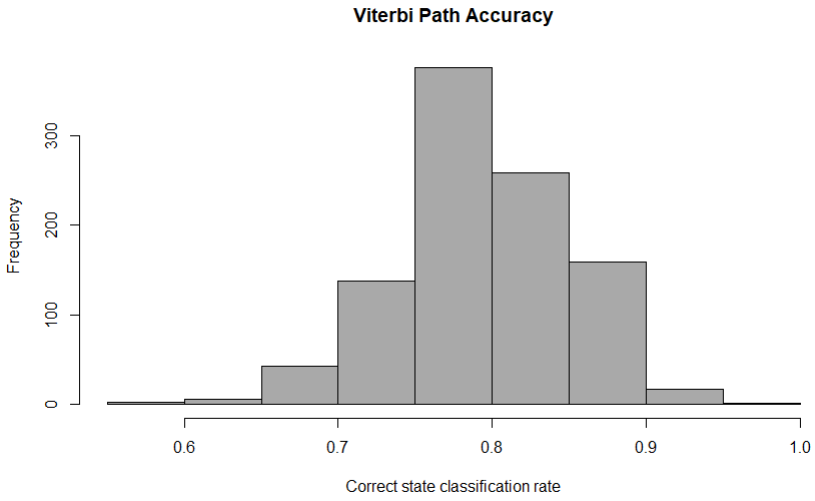


Figure 3.7: For a 3x3 state transition matrix T and a 3x3 emission matrix E with $n = 50$, where T have a diagonal of 0.4 and the other element of the matrix equals 0.3 while E have a diagonal of 0.8 and the other elements equals 0.1. Mean success of $79,906\%$

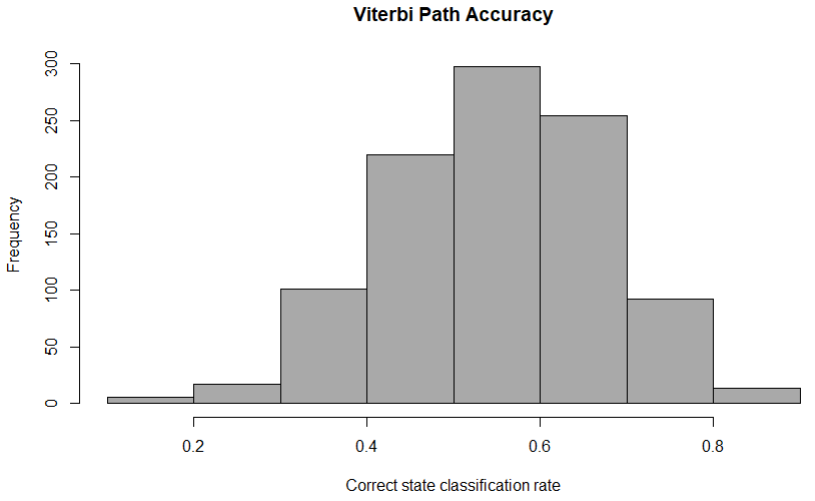


Figure 3.8: For a 4x4 state transition matrix T and a 4x4 emission matrix E with $n = 50$, where both T and E have a diagonal of 0.7 and the other element of the matrix equals 0.1. Mean success of $55,868\%$

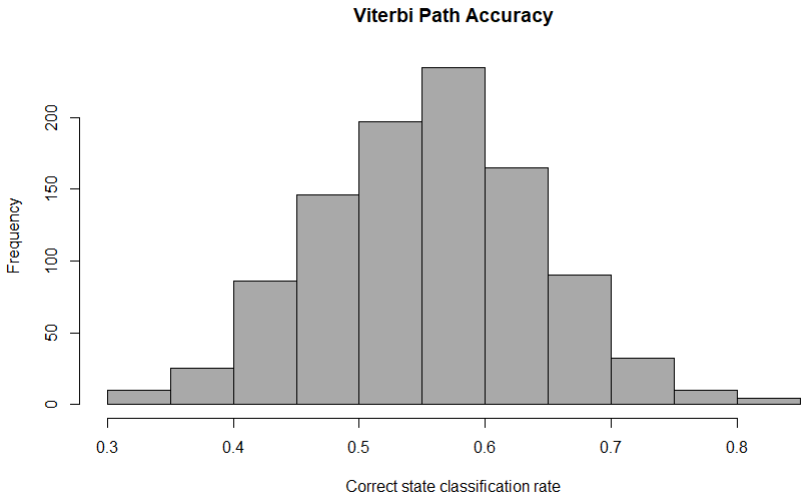


Figure 3.9: For a 4x4 state transition matrix T and a 4x4 emission matrix E with $n = 100$, where both T and E have a diagonal of 0.7 and the other element of the matrix equals 0.1. Mean success of $56,054\%$

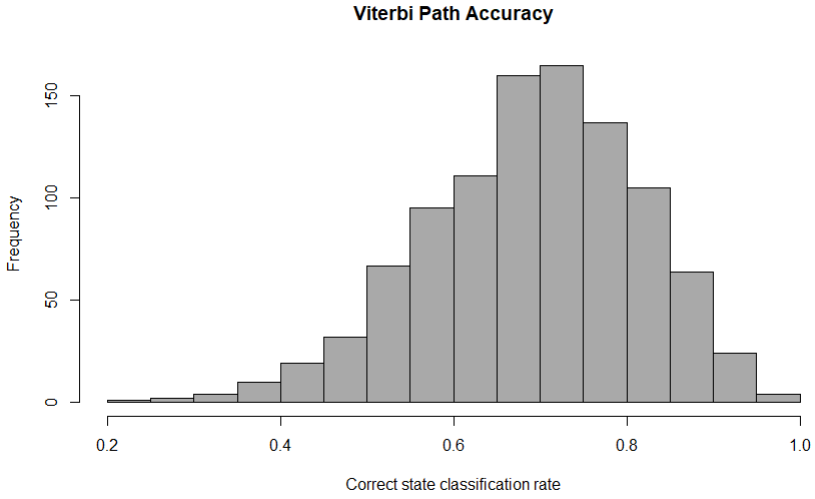


Figure 3.10: For a 4x4 state transition matrix T and a 4x4 emission matrix E with $n = 100$, where both T and E have a diagonal of 0.85 and the other element of the matrix equals 0.05. Mean success of $69,597\%$

very significantly when we change from $n = 50$ to $n = 100$, however the variance reduces drastically. If we compare variances from the case in Figure 3.8 and the case in Figure 3.9, the mean success variance reduces from 0.01414 to 0.00765. Thirdly, an increased state space and observation space gives an lower expected successful classification rate of the hidden states. None of these takeaways are very surprisings of nature, however the scale of them are interesting.

3.1.3 Extending Viterbi to forecasting

So far our only concern have been predicting hidden states of the past. In this section we will introduce two forecast algorithms. The first algorithm forecasts the next hidden state, while the second algorithm forecasts the observation Both of these algorithms are applied in Chapter 6.

Algorithm 2: Forecast Algorithm I

1) Run the Viterbi Algorithm

2) Solve $x^*_{T+1} := \arg\max(v[j,T] \cdot a_{ji})$, $i, j \in S$.

Result: Forecast x^*_{T+1} to be the next state.

Algorithm 3: Forecast Algorithm II

1) Run the Viterbi Algorithm

2) Solve $y^*_{T+1} := \arg\max(v[j,T] \cdot a_{ji} \cdot b_{ik})$, $i, j \in S$, $k \in O$.

Result: Forecast y^*_{T+1} to be the next observation.

The forecast algorithms makes use of the Viterbi path for $t = 1, ..., T$ and makes an prediction for $t = T + 1$ by combining the Viterbi calculations with the transition- and emission matrix of the HMM.

Chapter 4

Parameter estimation

This chapter revolves around estimating the underlying parameters of HMMs. Previously we have consider models where all the parameters of interest was given. This is however not always the case in real life problems. We need some techniques for instances where our information about the underlying processes are limited.

4.1 EM-algorithms

The Expectation-Maximization (EM) algorithm is a method to estimate unknown variables in a model. It contains of an *E-step* and a *M-step*. In the E-step, we make an initial guess of the model's parameters. We then obtain newly observed data. In the M-step, we fit these observations into our initial guesses and thus update the parameters of model. We have now return to the E-step and we are provided with more new observations that further updates our model to fit the data. This process is continued until we have converged to solution.

EM-algorithm will always improve, final model might only be a local maximum and not the global maximum. Therefore it is common to run an EM algorithm for several initial guesses and for there choose the one with largest likelihood as

guess for θ. In the next section, we will provide the mathematical properties for an special case of the EM-algorithm; the Baum-Welch algorithm.

4.2 Baum-Welch Algorithm

The Baum-Welch Algorithm is a special case of the EM-algorithm. This algorithm is used to estimate the unknown parameters of a HMM. It is sometimes referred to as the Forward-Backward Algorithm.

To proceed with the Baum-Welch algorithm, an observation sequence is needed. The goal is to find the underlying parameters that generate the observations. First, pick the starting values $\theta = (A, B, \pi)$, where A is the state transition matrix and B is the emission matrix. These values are the initial estimates of the HMM and can be selected at random, however, some prior information about the underlying model may give a more accurate initial guess which can have the potential find a better solution than purely random starting values. The Baum-Welch algorithm converges to a local optimum which might not be global optimum, hence, it is important to run this algorithm several times with different initial values to find the optimal estimate of the HMM.

Once θ is defined, the estimation of the HMM can start. The Baum-Welch algorithm contains of *forward calculation procedure* and a *backwards procedure*. The goal of the forward procedure is to calculate probability of being in state i at time t given the observation sequence and current estimate θ. The forward calculations defined as follows

$$\begin{aligned} &\alpha_i(t) = P(Y_1 = y_1, .., Y_t = y_t, X_t = i|\theta), \\ &\text{where } i \text{ is the current state at time } t, \end{aligned} \tag{4.1}$$

$$\alpha_i(1) = \pi_i b_i(y_1), \tag{4.2}$$

$$\alpha_i(t+1) = b_i(y_{t+1}) \sum_{j=1}^{N} \alpha_j(t) a_{ji}, \tag{4.3}$$

where a_{ji} is the current estimate of the probability of transition from state j to state i from one time point to the next and $b_i(y_t)$ is current estimate of the probability of observing y_t in state i.

The goal of the backwards procedure is to calculate the probability of observing the sequence $\mathbf{y} = (y_{t+1}, ..., y_T)$ when starting from state i at time t. The backwards calculations are defined as following

$$\begin{aligned} &\beta_i(t) = P(Y_{t+1} = y_{t+1}, ..., Y_T = y_T | X_t = i, \theta), \\ &\text{where } i \text{ is the current state at time } t, \end{aligned} \tag{4.4}$$

$$\beta_i(T) = 1, \tag{4.5}$$

$$\beta_i(t) = \sum_{j=1}^{N} \beta_j(t+1) a_{ij} b_j(y_{t+1}), \tag{4.6}$$

where $b_j(y_t)$ and a_{ij} follows the same definition as in the forward calculation.

After the calculations are done, the next objective is to update the parameters of the HMM. Thus, two new variables , $\gamma_i(t)$ and $\xi_{ij}(t)$ are created. $\gamma_i(t)$ is the new estimation of the probability of being in state i at time t, given the calculations done up to this point and is defined as follows

$$\gamma_i(t) = P(X_t = i | Y, \theta) = \frac{P(X_t = i, Y | \theta)}{P(Y | \theta)} = \frac{\alpha_i(t)\beta_i(t)}{\sum_{j=1}^{N} \alpha_j(t)\beta_j(t)}. \tag{4.7}$$

The other new variable, $\xi_{ij}(t)$, is the new estimation of the probability of going from state i to state j from on time point to the next given the calculations done in the forward- and backwards procedure, as well as the current estimate

of the HMM and is defined as follows

$$\xi_{ij}(t) = P(X_t = i, X_{t+1} = j|Y, \theta) = \frac{P(X_t = i, X_{t+1} = j, Y|\theta)}{P(Y|\theta)} = \frac{\alpha_i(t)a_{ij}\beta_j(t+1)b_j(y_{t+1})}{\sum_{j=1}^{N}\alpha_j(t)\beta_j(t)}. \tag{4.8}$$

The last step, after calculating $\gamma_i(t)$ and $\xi_{ij}(t)$, is updating the HMM parameters. Calculating the new estimates of the HMM as follows

$$\pi_i^* = \gamma_i(1), \tag{4.9}$$

$$a_{ij}^* = \frac{\sum_{t=1}^{T-1}\xi_{ij}(t)}{\sum_{t=1}^{T-1}\gamma_i(t)}, \tag{4.10}$$

$$\begin{aligned} &b_i^*(o_k) = \frac{\sum_{t=1}^{T} 1_{y_t=o_k}\gamma_i(t)}{\sum_{t=1}^{T}\gamma_i(t)}, \\ &\text{where } 1_{y_t=o_k} = 1 \text{ if } y_t = o_k, 0 \text{ otherwise.} \end{aligned} \tag{4.11}$$

π^* is the new estimate of the stationary distribution, where π_i^* is the estimated relative frequency of time spent in state i. a_{ij}^* is the new estimated of the transition probability from state i to j. $b_i^*(v_k)$ is the new estimate of the probability of seeing the observation o_k in state i, $o_k \in O$ where O is the observation space.

This process is repeated until the algorithm have converged to a solution. Convergence is determined by a set tolerance value δ. The difference of transition and emission parameters must be smaller than δ to have reached convergence and thus terminate the algorithm.

Algorithm 4: Baum-Welch Algorithm

Initial values: State space $S = (s_1, s_2, ..., s_K)$, observation space $O = (o_1, o_2, ..., o_N)$, initial guess $\theta = (A, B, \pi)$, a tolerance δ and the observed sequence Y.

Set $A = A^*, B = B^*, \pi = \pi^*$.

while *change in parameters* $\leq \delta$ **do**

$\alpha_i(1) \leftarrow \pi_i^* b_i^*(y_1)$

$\alpha_i(t+1) \leftarrow b_i^*(y_{t+1}) \sum_{j=1}^{N} \alpha_j(t) a_{ji}^*$

$\beta_i(T) \leftarrow 1$

$\beta_i(t) \leftarrow \sum_{j+1}^{N} \beta_j(t+1) a_{ij}^* b_j^*(y_{t+1})$

$\gamma_i(t) \leftarrow \frac{\alpha_i(t)\beta_i(t)}{\sum_{j=1}^{N} \alpha_j(t)\beta_j(t)}$

$\xi_{ij} \leftarrow \frac{\alpha_i(t) a_{ij}^* \beta_j(t+1) b_j^*(y_{t+1})}{\sum_{j=1}^{N} \alpha_j(t)\beta_j(t)}$

$\pi_i^* \leftarrow \gamma_i(1)$

$a_{ij}^* \leftarrow \frac{\sum_{t=1}^{T-1} \xi_{ij}(t)}{\sum_{t=1}^{T-1} \gamma_i(t)}$

$b_i^*(o_k) \leftarrow \frac{\sum_{t=1}^{T} 1_{y_t = o_k} \gamma_i(t)}{\sum_{t=1}^{T} \gamma_i(t)}$

end

Result: *Local optimum* $\theta^* = (A^*, B^*, \pi^*)$

Chapter 5

A Brief Introduction to the Stock Market and Financial Data

Before we apply HMM to a stock market trading algorithm, we need to give a short introduction financial data and the stock market.

5.1 Time Series

5.1.1 Short introduction to Time Series

A time series is a collection of observations indexed by time [8]. The only requirement for a collection of observations to be considered a time series is that the collection of observations are of the same phenomenon in the same environment over time. As an example, an observation sequence of the mean temperature in London over the last 30 days meets this requirement and can be considered a time series. If you change the city each day, it would no longer be considered a time series, rather a collection of 30 different observations not

observing the same phenomenon.

Time series can be classified into two different types; stock- and flow time series [9]. The difference between the two classes is in the method of measuring a phenomenon.

Stock time series := Collections observations at distinct times.
Flow time series := Collects observations continuously throughout a given time frame.

A time series is usually broken down into three components; *trend*, *season* and *irregularity* [9]. The trend is the long term trajectory of the time series. The season of a time series is the systematic behaviour related to a calendar event (time of year, time of month, time of day etc.)[9]. The irregularity component is the noisy short term fluctuations of the time series. Two typical models of an observed time series (which are just two of many possible relationships between the components) are given by [9].

$$\begin{aligned} &\text{Observed Series} = \text{Trend} + \text{Season} + \text{Irregularity}, \\ &O_t = T_t + S_t + I_t. \end{aligned} \tag{5.1}$$

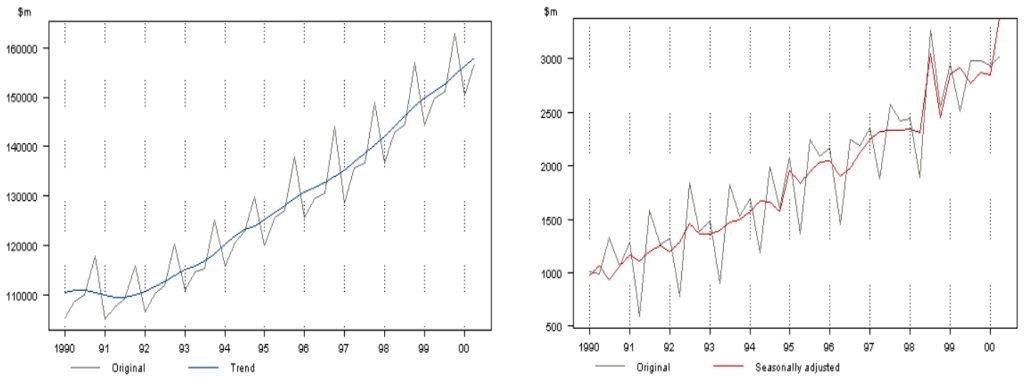
$$\begin{aligned} &\text{Observed series} = \text{Trend} \cdot \text{Season} \cdot \text{Irregularity}, \\ &O_t = T_t \cdot S_t \cdot I_t. \end{aligned} \tag{5.2}$$

Hence, we can also rewrite (5.1) or (5.2) if we want to explore one of the components in the relevant model. A seasonal adjusted model on form of (5.1) can be written as

$$\begin{aligned} &\text{Seasonally adjusted series} = \text{Observed series} - \text{Seasonal Component} \\ &= \text{Trend} + \text{Irregularity}, \\ &SA_t = O_t - \hat{S}_t = T_t + I_t, \end{aligned} \tag{5.3}$$

where $\hat{S}_t$ is the observed seasonality [9]. Not all time series have a seasonal component, which corresponds to $S_t = 0$ in equation (5.1), hence

$$\text{No seasonal component} \longrightarrow O_t = T_t + I_t. \tag{5.4}$$



(a) A trend component of an observed series and the corresponding series.

(b) A time series with corresponding season adjusted model.

Figure 5.1

5.1.2 Stationary time series

Stationarity is an important concept in time series. Let X_t be a time series and define

$$\mu_X(t) := \textbf{the mean function} \text{ of } X_t,$$
$$\gamma_X(r, s) := \textbf{the covariance function} \text{ of } X_t,$$
where r and s are time points.

A time series X_t is (weakly) stationary if $\mu_X(t)$ is independent of t and if $\gamma_X(t + h, t)$ is independent of t for every h [10]. When we talk about a time series being stationary in this thesis, we always refer to the weakly definition. In other words, a stationary series is a time series with a neutral trend component and without a seasonal component, making the mean value independent of time.

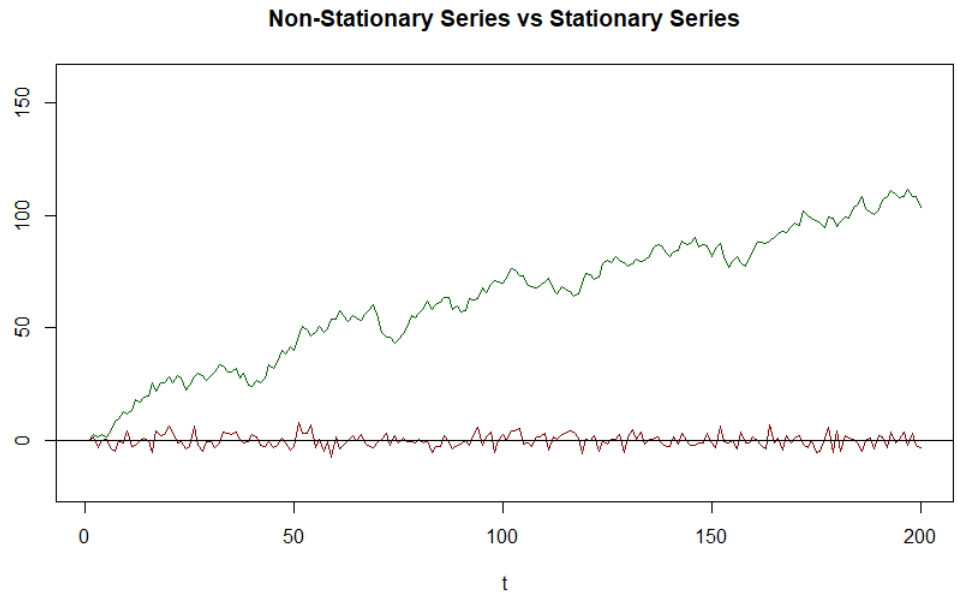
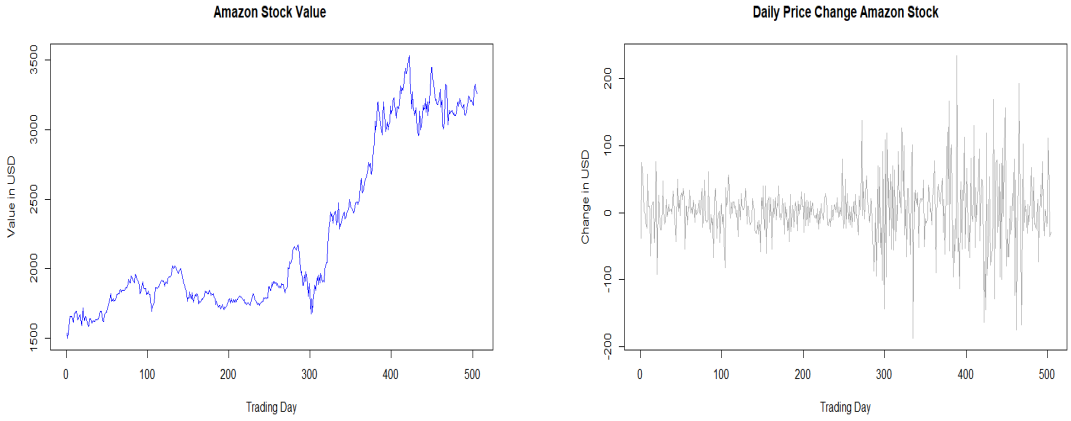


Figure 5.2: Illustration of a stationary time series (red) and non-stationary time series (green)

There plenty of material available on how to model both stationary and non-stationary times in a "traditional" manner, however that is outside the scope of this thesis. In chapter 6 we introduce our approach to model a (financial) time series using stochastic modeling.

5.1.3 Financial time series

The term *financial time series* refers to an observed series of a financial quantity. Some examples would be daily/monthly/yearly revenue of a company, changes in oil price etc. In this thesis however, we are mainly concerned with financial time series observing phenomenons in the stock market.



(a) (b)

Figure 5.3: The Amazon stock value from 01 January 2019 to 01 January 2021.

5.2 Stock Market

5.2.1 Introduction

A *stock* is a security that represents the ownership of a fraction of a corporation [12]. These entities are first sold buy corporation itself, often to raise capital, growing the business, pay of depth etc [13]. Stocks can be traded privately,

although they are most commonly traded in the *stock market*.

After stocks are sold by the corporation, stocks on the stock market can be bought and sold continuously given that a buyer and a seller can agree on a price. Often these tradings are done by *stockbrokers*. You request a price for each of the stocks you are buying/selling and the stockbroker will complete the transfer when a seller/buyer is willing to trade at the requested price. There are many platforms available to trade stocks. However, there is an expense related to trading stocks, which is the *commission*. Commission is a fraction buying/selling price of the stocks traded which differs depending on which platform you use. As an example, the platform "Nordnet" normally takes a $0,049\%$ commission of the buying/selling price when trading stocks within the Nordic countries.

So far we have introduced what a stock is, but what drives the stock prize? There are techniques to evaluate what a stock should be worth, by assessing a company's earnings and expenditures. Ultimately however, the actual stock prize is determined by the supply and demand on that stock at that given time in the market[14].

A *stock index* measures the stock market by taking a subset of stocks in market and observe them over time[15]. Some indexes take a subset of stocks from the entire market (S&P500, Dow Jones Industrial Average etc.), other look take subsets from a given country (OSEBX etc.) and some indexes look specifically on a subset of stocks from given industries (NASDAQ Biotechnology Index etc.). Historically, the stock market have on average continued to raise in value with time, although there have been some down periods at certain time intervals.

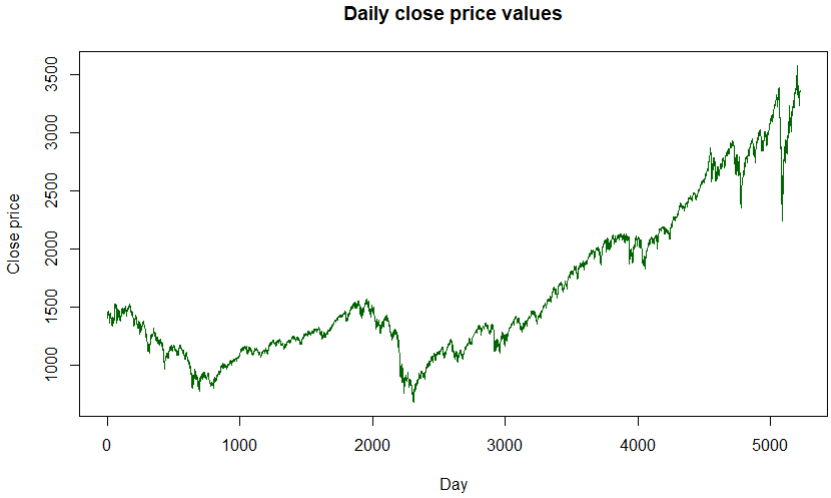


Figure 5.4: S&P500 Index over the last 20 years. Days is open trading days.

5.2.2 Bull- and Bear market

Although the stock market showed a positive trend throughout time, there are quite a few periods where the stock market have seen a negative trend. Times of negative and positive trends are commonly separated into *bull market* and *bear market*.

Bull Market := The stock market have an overall positive trend.

Bear Market := The stock market sees an overall decreasing trend.

A bull market is often a case of investors showing great confidence and trading volumes are high. On the flip side, a bear market is usually the case of investors being quite pessimistic about the near economical future, stock prizes falls or stagnates and overall trading volumes are low[16].

5.2.3 Investment strategies

For every individual trading stocks, not matter the scale, it is useful to have a specific trading strategy, or at the very least have an basic understanding of the nature of stock prizes. The simplest strategy is simply to *buy and hold* a diversified *portfolio*. Your portfolio is simply put your collection of stocks. The reason for diversifying your portfolio is to minimize the risk associated with investing in stocks. We can never perfectly predict the future of a stock and any stock can see an unexpected turn for the worse not matter how bright we anticipates its future to be. Combining a number of stocks in your portfolio will decrease this risk, especially when they are diversified in different industries [17].

Another way to run the buy-and-hold strategy is to diversify your portfolio in one specific industry. This is associated with more risk due to the fact that stocks in the same branches tend to be bought and sold for much of the same reason, namely that investors believe in that particular industry. Although more risky, this approach may lead to a larger profit when a certain industry succeeds. As an example, the fund "DNB Teknologi A" which mainly diversifies stocks in the technology branch, have seen over 500% return over the last ten year. This is significantly more than more the average return, measured by the S&P500 Index, in the same time frame.

We can choose to take an more active approach compared to the buy-and-hold approach by investing in stocks for a shorter time period and try to sell them at the most profitable time. There is also the option to combine these strategies of course, buying some stocks for the long term and others trying to make a short term profit. Two usual approaches to take when buying a stock is refer to as taking a *Long Position* or taking a *Short Position*[2].

Long Position := Buy an asset hoping for an increase in price.
Short Position := Buy an asset hoping for a decrease in price.

It is quite intuitive how the long position is profitable when successfully executed. The short position is profitable we buy stocks falling in value and then other stockholders start buying more stocks to cover some of their average return per individual stock. If successful, this "panic" behaviour leads to the stock increasing in value again and hence a profit is made.

Following an active investment strategy I think most will agree it requires a good understanding of the stock market and underlying factors that in part drives stock prizes, a good chunk of time and a some degree of luck to go with it to succeed. In the next chapter we will look at a case study applying stochastic modeling in an investment strategy using discrete hidden Markov models.

Chapter 6

Application of HMMs in stock trading algorithms

This chapther is a case study in the usage of HMM-based stock trading algorithms. We will introduce two candidate algorithms, named Algorithm I and Algorithm II respectively. The work is influenced by the model in [2] but made from scratch. Algorithm I is a close remake of the model in [2], while Algorithm II have been modified to contain four states as opposed to three. Furthermore, Algorithm II is constructed to examine the effects of defining the hidden states as stock trading signals, rather than making predetermined signals based on the forecast of the next day close prize behaviour.

6.1 Our Data

In this case study we will use the S&P500 index as our "stock" of interest. The S&P500 index includes over 500 of the leading companies in world economy and is regarded as one of the most popular indexes in the stock market [3].

Although the S&P500 index is not a stock, rather a multitude of stocks, we

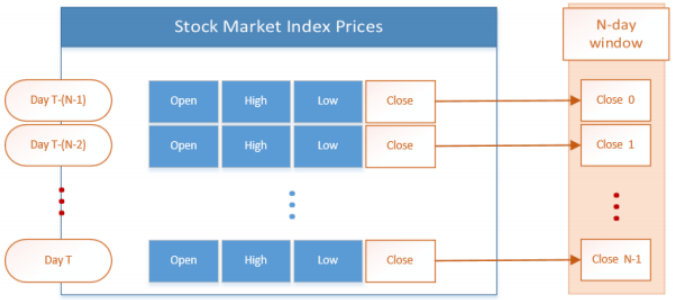
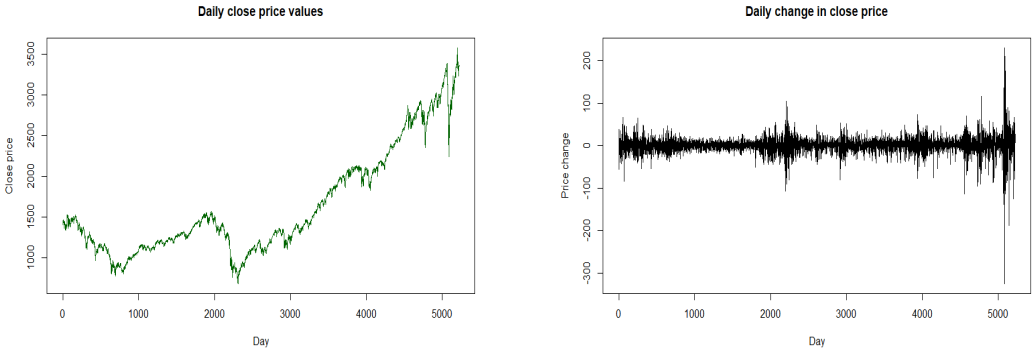


Figure 6.1: Illustrating the extraction of close prices from the SP500 index. Illustration taken from [2]

will assume that it behaves like a stock in this case study. We will only evaluate the closing prices and exclude the other values (open price, highest price, lowest price and volume). The data was downloaded from [19] and only consider days were stock exchanges are done.



(a) (b)

Figure 6.2: Image (a) illustrates how the S&P500 closing price have evolved from 03.01.2000 to 30.09.2020. Image (b) shows the change in close price from the previous day in the same time period.

Figure 6.2 is an illustration of the S&P500 index closing price over the last 20 years. Figure 6.2 (a) shows that before the financial crisis [6] hit global economy, the closing price seems to be oscillating around a value in the 1200-1300 range

and in Figure 6.2 (b) that the daily changes seems to have a decreasing trend. Since the financial crisis however, the closing price have seen a tremendous increase with very few periods of negative trend. Finally, we can see the breakout of the corona pandemic destabilized the closing price the most (measured in daily total change, not necessarily in relative change), where the closing price dropped drastically and then lately surpass the pre-pandemic values.

In this chapter we will create two separate trading algorithms using HMMs, as mentioned in the introduction of this chapter. All of these HMMs will be trained in the time period from 10.01.2003 to 09.01.2007 and then tested from 12.01.2009 to 12.01.2017, following the training- and test periods from [2]. We will test the two algorithms by comparing their achieved *rate-of-return* (ROR) over the testing period. The results will be compared to the ROR achieved by a passive buy-and-hold strategy, which buys one unit of the stock and then keeps it throughout the testing period.

6.2 HMM architecture

In this section we will discuss the setup of our HMMs. The HMMs role in this trading algorithm is to create an (hopefully good) understanding how the SP500 index closing prices are driven.

6.2.1 Transforming Observations

The stock market trades in continuous values and one may think that it would make sense to pursue the task of predicting stock prizes in a continuous manner. When we try to forecast stock prizes using continuous observations, the task becomes to predict the exact prize the next day. If we follow an approach using discrete observations, we change the task from predicting the stock prize

to predicting the direction of the stock prize, whether it raises, maintains or falls. We can split the observations type further by splitting raises/falls in price into strong/weak/moderate categories as well as choosing how strict/loose we want to define the maintenance term.

As briefly mentioned above, we have some options how to categorize the close prize observations. In this thesis we will rely on the study made in [2], which suggests that using three types of observations yields the best results. The observations are separated into three categories, "Rise", "Decrease" and "Strict Maintenance". We define P_t = closing price at trading day t. We separate the observations into categories the follow way

$$\begin{aligned}
&\text{Rise } \leftarrow P_{t+1} - P_t > 0, \\
&\text{Decrease } \leftarrow P_{t+1} - P_t < 0, \\
&\text{Strict Maintenance } \leftarrow P_{t+1} - P_t = 0.
\end{aligned}$$

We have only considered the use of a "strict" maintenance term in this thesis based on the study in of observation types in [2]. Only using strict maintenance leads to very few observations in this category and could be considered as an excessive category, however, we do not have to worry about categorizing static behaviour as either a "rise" or a "decrease".

6.2.2 Defining the Hidden States

Here, we present two experiments. Algorithm I will use three hidden states, similar to the model in [2], while Algorithm II will use four hidden states. We will now justify the approach taken in Algorithm II.

The definition of hidden states are inherently difficult and there is no clear rules how to do so. In fact, many approaches have tried with varying results

[2].That being said, when I first did some research for this thesis looking attempts of implementing HMMs in stock trading algorithms and read through [2] for the first time, my intuitive thought was that the hidden states was the true state of the stock in the sense that the state had an concrete action associated with it. In the stock market, signals are often given by stock trading experts based on their assumptions on how a given stock prize will evolve for a period of time. In Algorithm II, we define the states as trading signals, namely "Strong Buy", "Hold", "Sell" and "Strong Sell". The potential benefit with this approach is that we have some previous knowledge before we train the HMMs using the Baum-Welch algorithm. To illustrate this statement, it would not be far fetched to suggest that a "Strong Buy" state would indicate a positive trend, "Hold" a steady trend, "Sell" and "Strong Sell" would indicate negative trends. These four states was chosen because these are the exact signals used in Algorithm I. We will elaborate on this in Section 6.3.

6.2.3 Estimating hidden states and forecasting

In this thesis we use the Viterbi Algorithm to estimate the hidden states, which was introduced in Chapter 3. The estimations achieved by the Viterbi Algorithm will then be used in the calculations leading to the forecast.

Using the Viterbi Algorithm, we have to provide an string of input observations. This leads to the question; can we find an optimal input window that maximizes the probability of obtaining the right state at the current time? Estimating the right state at the current time would thus lead to the best prediction for the next closing price direction and the best prediction for the next state. In Section 3.1.2 we had a brief look at this question. When we compared HMMs which only differed in the length of the observation sequences (one with length of 50, the other with a length of 100), the one with the longer observation sequence had only marginal better correct estimation rate. This study was of course very limited and the HMM parameters was quite different than the ones

Window Size (days)	ROR	Error	Sharpe
5	2,9%	51%	0.2
15	15,7%	51%	0.7
30	26,6%	49%	1.10
40	25,9%	48%	1.09
50	20,8%	47%	0.92
60	36,1%	46%	1.44
75	16.0%	48%	0.73
90	0,7%	48%	0.1
180	-2,4%	48%	-0.04

Figure 6.3: Case study done in [2] showing rate-of-return, prediction error and sharpe ratio for different number of observation sequence lengths in DHMMs (referred to as Window Sizes).

we are dealing with in this chapter, thus we rely on the case study in [2], which is summarized in Figure 6.3. As seen in Figure 6.3, a 60-day window gives the highest rate of return (ROR) according to [2], with 30-, 40- and 50-day windows also producing RORs over 20%. In these experiments, we will therefore consider HMMs with 30 and 60 observation inputs. Why exactly these two sizes will be explained in the next section when we introduce the usage of multiple HMMs.

An improved estimate of the hidden states, ultimately leads forecast. However, the Viterbi Algorithm is only concerned with estimating the past and the present. For Algorithm II, we want to forecast the next state which is then used as a signal for adjusting/maintaining our market position (see Section 6.3). This is done by following Forecast Algorithm I from Section 3.1.3. For Algorithm I, we want to forecast next observation rather than the next state. This is done by following the steps of Forecast Algorithm II from Section 3.1.3.

6.2.4 Multiple HMMs

We have previously discussed the number of states and number of observations that we will be using in our stock trading experiments. Now we will introduce another layer of complexity to our algorithm; multiple HMMs. Following a case study done in [2], using two daily HMMs with a 30 and 60 observation window in

Window Sizes (days)	ROR	Error	Sharpe
15, 30	5.0%	50%	0.29
15, 40	17.0%	49%	0.76
15, 50	16.0%	49%	0.71
15, 60	28.2%	48%	1.13
15, 75	8.9%	49%	0.43
30, 40	29.2%	48%	1.12
30, 50	23.8%	48%	1.02
30, 60	30.1%	47%	1.24
30, 75	17.6%	48%	0.79
40, 50	19.3%	47%	0.86
40, 60	20.2%	47%	0.89
40, 75	10.8%	47%	0.52
50, 60	22.4%	47%	0.97
50, 75	15.4%	47%	0.71
60, 75	20.4%	47%	0.90

Figure 6.4: Case study done in [2] showing rate-of-return, prediction error and sharpe ratio for double daily DHMMs using different observation lengths (referred to as Window Sizes).

conjunction with one weekly HMM with a 60 week observation window seemed to yield the best results.

The goal of using multiple DHMMs is to further improve the estimation and forecast. For the two daily HMMs, we only consider a prediction valid if the 30-day HMM and the 60-day HMM gives an unanimous prediction. In addition to the two daily HMMs, there is also the weekly HMM, using a 30-week window. This model used in [2] and we will adopt this strategy. Thus, the predictions are either made by the two daily HMMs or weekly HMM. Following the model in [2], we will use *Relative Strength Index* (RSI) to decide which of HMMs are being used. The RSI value is calculated using the values *Average Loss* (AL) and *Average Gain* (AG). AL and AG are defined as follows

$$\text{AG} = \frac{\text{sum of price increases over time period}}{\text{Time Period}}, \tag{6.1}$$

$$\text{AL} = \frac{\text{sum of price decreases over time period}}{\text{Time Period}}. \tag{6.2}$$

The RSI calculation then follows

$$RSI = 100 - \frac{100}{1 + \frac{AG}{AL}}. \tag{6.3}$$

A stock is consider overbought when the RSI value surpasses 70 and likewise oversold when the RSI value is reaches below 30 [2][7]. An overbought stock is viewed as being overvalued and may experience a pullback in price soon, while an oversold stock is considered an undervalued asset and might experience a positive trend in the near future [2]. The RSI value can be calculated using different time periods, however, a 14 day period is commonly used. We will follow the standard approach in this thesis, setting the time period to 14 days [7].

The RSI value is applied in our thesis to give a criteria for when to switch between the weekly and daily HMMs. This is entirely motivated by a case study in [2], comparing three different technical indicators and the performance of the HMMs using each of these. In our experiments, we start using the the two daily HMMs by default. We use the daily HMMs until we have a RSI value below 30. When this occurs, we switch over to the weekly HMM and continue using it until get an RSI value above 70. If that is the case, we switch back to the daily HMMs and repeat this evaluation process continuously through the experiment period.

6.2.5 Training the HMMs

In this study, train the HMMs using Baum-Welch algorithm. This is done for the HMMs in Algorithm I by choosing some random values. Remember, we do not attempt to interpret the states in Algorithm I. For Algorithm II, we choose

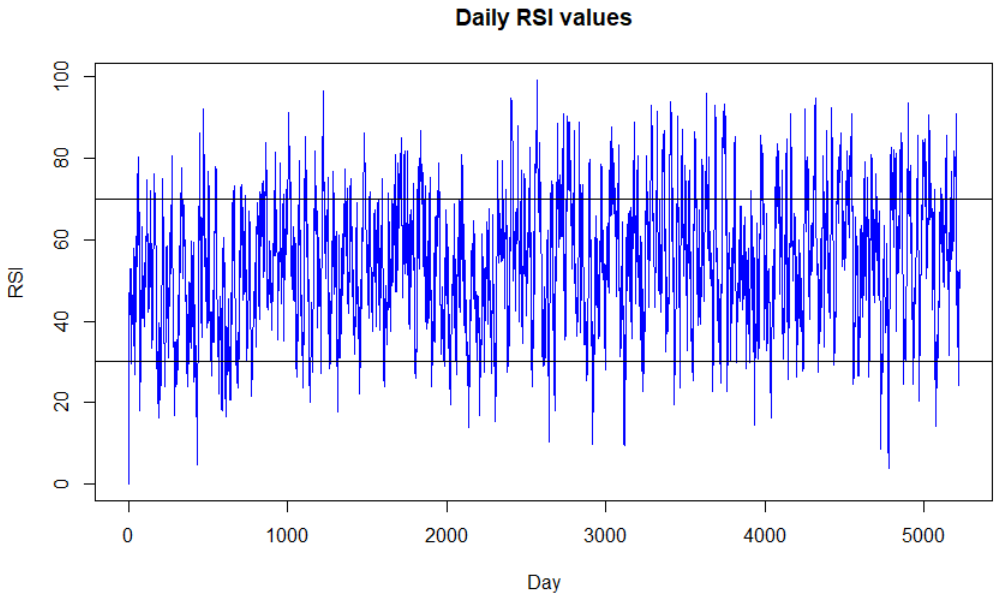


Figure 6.5: Daily RSI values from 03.01.2000 to 30.09.2020 calculated and generated from R. The two horizontal lines is set at the values 30 and 70.

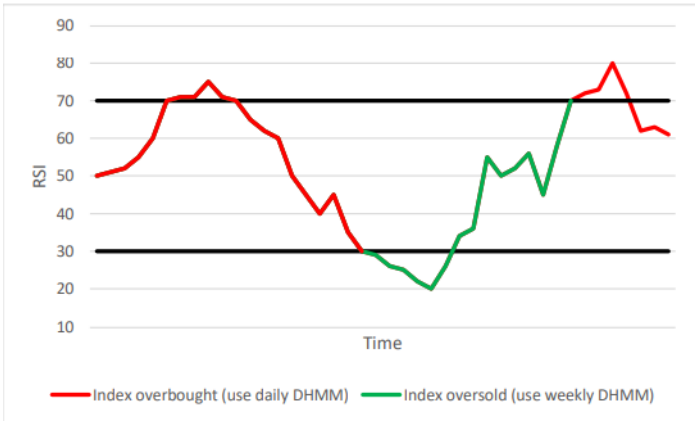


Figure 6.6: Illustrating the process of choosing which DHMM to use. Illustration credit to L. Andrade (2017)[2]

values which seems to make sense for the different states. Here, we have defined the states as "Strong Buy", "Hold", "Sell" and "Strong Sell". We then evaluate the results and repeat the Baum-Welch algorithm with new starting values until we have achieved desirable estimates for the HMMs. We used the the function 'baumWelch()' for the R-package 'HMM'[18] to do these calculations.

6.3 Decision Algorithm

In this section we are going to show how we integrate the HMMs into a complete stock trading algorithms, namely Algorithm I and Algorithm II.

Following the algorithm in [2], we consider three different positions in the market. The three positions are *out-of-market*, *long position* and *short position*, which were all defined in chapter 5. Both when we hold a long position and a short position, we are in the market which means we are invested in the stock. In this thesis we will only buy one unit of the stock each we enter the market. When we leave the market, we sell this unit.

Before we describe how me move between the different positions, we have to clarify the signals given by the prediction core in Algorithm I. The HMMs in conjunction with the Viterbi algorithm makes an forecast of the next price direction, following Forecast Algorithm II from Section 3.1.3. Based on the forecast, one of the four following signals are given; "Strong Buy", "Hold", "Sell", "Strong Sell". These signals are made based on the forecast the following way

Strong Buy $\longleftarrow$ A "Rise" prediction is made.
Strong Sell $\longleftarrow$ A "Decrease" prediction is made using daily DHMMs.
Sell $\longleftarrow$ A "Decrease" prediction is made using weekly DHMM.
Hold $\longleftarrow$ Using the daily DHMMs, an inconclusive prediction is made.

Algorithm II interpreter the hidden states as signals, thus we forecast the state

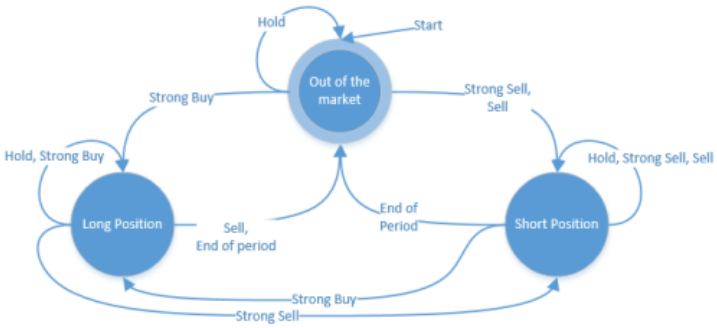


Figure 6.7: The investment process. Illustration credits to L. Andrade (2017)[2].

rather than the observation when working with that algorithm. The forecast is made using Forecast Algorithm I from Section 3.1.3. The next state prediction is regarded as the signal and we adjust our position accordingly.

We always start our investment period out-of-market. We stay there until a prediction other than "Hold" is made. When the prediction "Strong Buy" is made, we switch to a long position. If a "Sell" or "Strong Sell" prediction is made, we move to a short position. When we hold a short position, we stay there until we get a "Strong Buy" prediction and then we move to a long position. In a long position, we adjust to a short position when the prediction "Strong Sell" is made and we move out of market if we get an "Sell" prediction. A "Strong Buy" or a "Hold" signal has a neutral effect when in the long position. This all following the model in [2]. A diagram illustrating the investment strategy is given in Figure 6.6.

6.4 Results

In this section we provide the empirical results of our stock trading algorithms along with our interpretations of the results.

6.4.1 Overall results for testing period

The results shows that we were not able to reproduce the exact results in [2] with our similar algorithm named Algorithm I. However, Algorithm I still achieved a slightly better ROR than the buy-and-hold strategy. Our other algorithm, Algorithm II, were able to outperform both Algorithm I and the buy-and-hold strategy by a substantial margin. Below, the empirical results will be given in the form of plots and tables. In addition, some interpretations and remarks of the data will be provided.

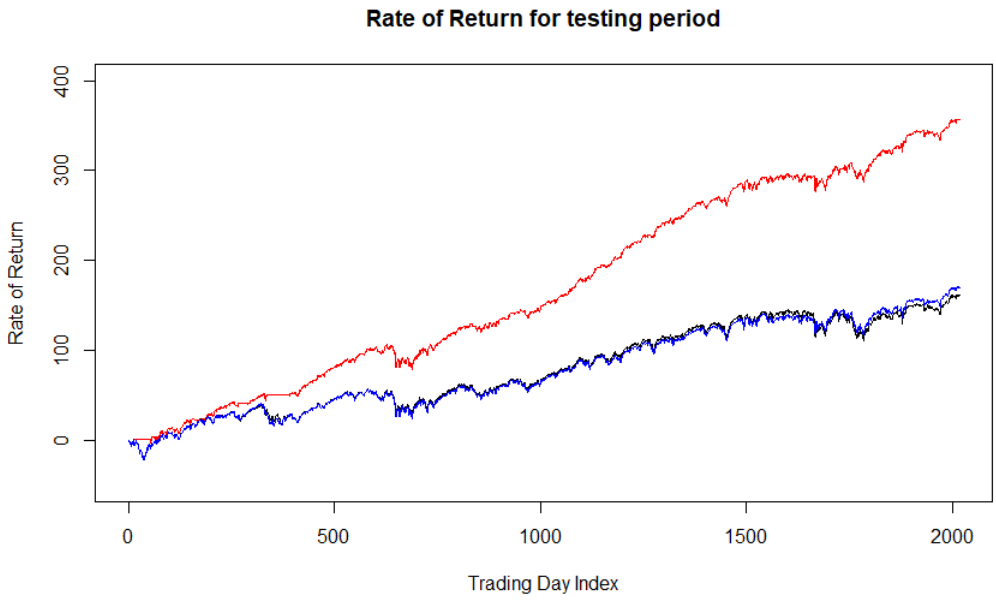
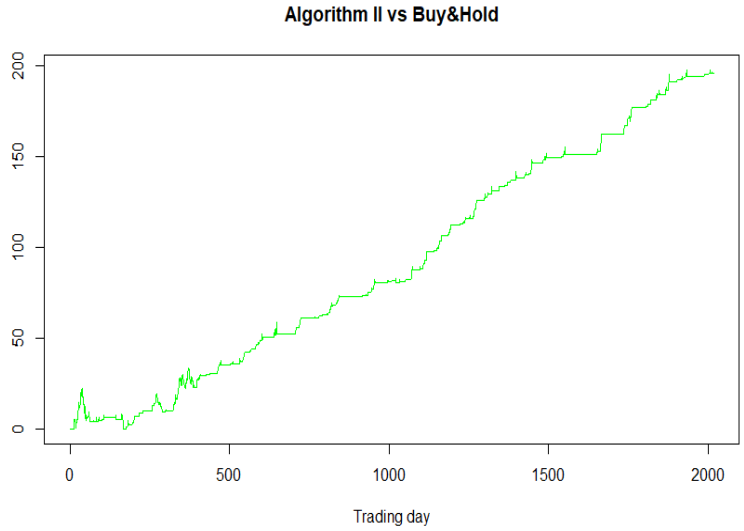
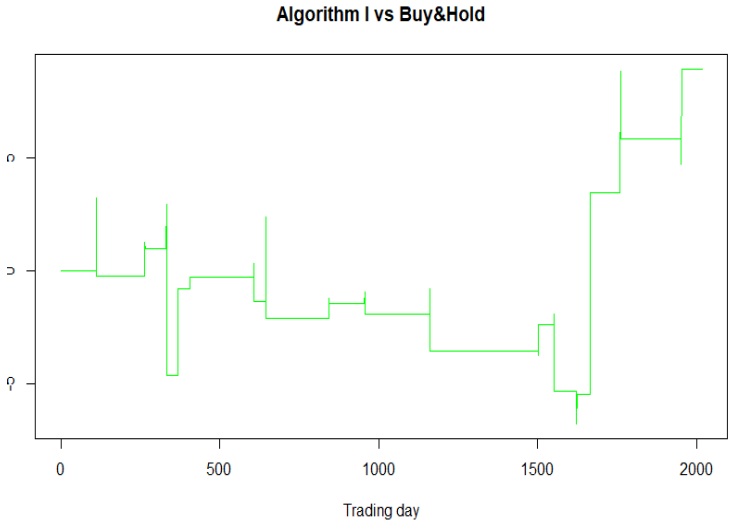


Figure 6.8: Rate of return for testing period. The black line is the buy-and-hold strategy, the blue line is Algoirthm I and the red line represents Algorithm II.

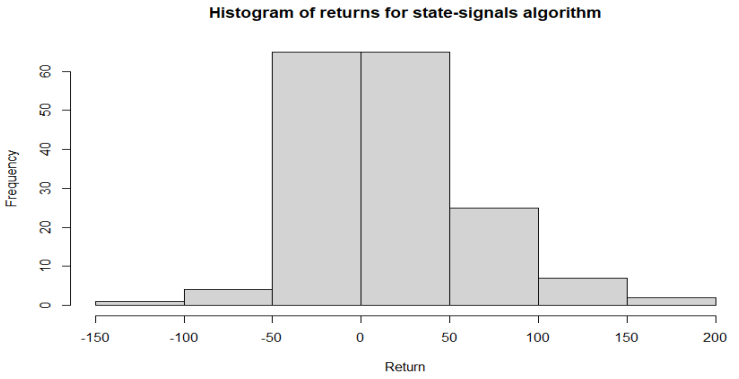


(a)

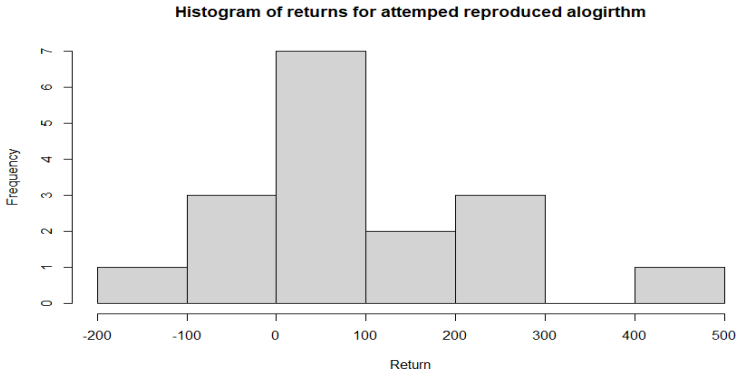


(b)

Figure 6.9: Evolution of ROR achieved by Algorithm II (a) and Algorithm I (b), relative to the ROR of the Buy-and-Hold strategy.



(a)



(b)

Figure 6.10: Size of individual returns made by Algorithm II (a) and Algorithm I (b).

	Buy-and-hold	**Algorithm I**	**Algorithm II**
Overall ROR	$160,89\%$	$169,85\%$	$356,62\%$
Days out of market	0	17	296
Days in short position	0	338	17
Days in long position	2016	1661	1703

Table 6.1: Overall ROR and positions taken on trading days

An interesting result of this study, is the fact that usage hidden states as signal seems to have merit in stock trading algorithms based on the empirical data in this study. Algorithm II found an active approach to outperform both Algorithm I, using predefined signals, and the passive buy-and-hold strategy. Although the empirical data heavily suggests using hidden states as signals rather than using predefined signals, we have to remember that [2] achieved a great ROR for the same time period and the same "stock" used in our study. Hence, we might not have converged to the global optimum in Algorithm I. In Table 6.1, it is clear to see that Algorithm II converged to a solution taking a more active approach than Algorithm I. This is evident when inspecting Figure 6.9.

	Algorithm I	**Algorithm II**
"Strong Buy" signals	1649	1058
"Hold" signals	13	773
"Sell" signals	222	185
"Strong Sell" signals	132	0

Table 6.2: Showcasing the number signals made for both algorithms.

Table 6.2 provides answers for the positions taken during the test period for our two algorithms. Algorithm I produced more "Sell" signals than Algorithm II, which perhaps sounds counter intuitive given that the "Sell" signal is the only signal that moves the algorithms from an in-market position (long- or short position) to an out-of-market position. This suggests that the vast majority of the

"Sell" signal given by the prediction core in Algorithm I came in a short position. Furthermore, Algorithm I did only receive 13 "Hold" signals. Compared to the 773 "Hold" signal given in Algorithm II, it seems like the "Hold" signal is an important piece of the puzzle to obtain a high ROR.

	Algorithm I	**Algorithm II**
No decision made	13	4
Prediction error	$45,39\%$	-

Table 6.3: Number of days where no decision was made due to the 30-day HMM and the 60-day HMM made different forecasts. This table also include the prediction error of Algorithm I. Remark that there is no prediction error listed for Algortihm II, because we have no way know the true hidden states.

Elaborating further, the "Hold" signal is given in Algorithm I when the two DHMMs makes two different forecast of the closing price direction. We can see that even though Algorithm I only failed to make a forecast 13, Algorithm II only encountered an inconclusive prediction 4 times. Keep in mind, Algorithm I predicts the next close prize direction while Algorithm II predicts the next hidden states, where the hidden states are defined to be the signal output which dictates the next action done by the algorithm. This may suggest that the definition of producing "Hold" signals in Algorithm I is sub optimal. Inspecting Table 6.2 further, it seems that less frequent "Strong Buy" signal is beneficial.

We are yet to comment on the lack of "Strong Sell" signal given in Algorithm II. In this thesis, our main goal is to showcase how using the hidden states as trading signals performed, versus the signal strategy using close prize directions and predefined actions given these signals inspired by [2]. Thus, we stuck with a framework of four hidden states in Algorithm II. Given the fact Algorithm II gave zero "Strong Sell" signals, suggest two things. First, having two signals related to a negative close price directions might not be necessary. Having only one signal related to positive close prize directions, might have played a part in this. Secondly, four states might not be optimal for modeling stock prize

behaviour. Algorithm II achieved a high ROR compared to Algorithm I and the buy-and-hold strategy, however, there can potentially be a way to define the hidden state which would have been even more precise. Of course, one could argue the other way and suggest that the "Strong Sell" simply did not occur in the test period, given positive trend throughout, as showcased in Figure 6.2 (a). All we can do for now is speculate and motivate for further studying of this topic.

6.4.2 Estimated HMMs

For recreational purposes, the estimated HMMs for both algorithms displayed in this subsection. We obtained these solutions by trying numerous different starting values. We obtained a lot of solutions sub optimal to the ones showcased in this chapter. Throughout this processes of modifying the starting values, we observed that using a tolerance level $\delta = 0.005$ produced the estimates which achieved the highest rate of return. This suggest that $\delta = 0.005$ is close to an optimal value in the bias/flexibility dilemma, where the more flexible a model is makes it adjust more to the training values and the more biased a model is makes it adjust less to the training values. We conclude this chapter by providing the estimated HMMs. Transition matrices are denoted $T_{z,i}$ and emission matrices are denoted $E_{z,i}$. $z = (D, W)$, D when a daily HMM and W when weekly, and $i = (1, 2)$, where $i = 1$ represents Algorithm I and $i = 2$ represents Algorithm II

$$T_{D,1} = \begin{array}{c} \\ Z_1 \\ Z_2 \\ Z_3 \end{array} \begin{array}{c} \begin{array}{ccc} Z_1 & Z_2 & Z_3 \end{array} \\ \begin{bmatrix} 0.1303045 & 0.6236142 & 0.2460813 \\ 0.3841310 & 0.4297123 & 0.1861567 \\ 0.1277891 & 0.2114254 & 0.6607855 \end{bmatrix} \end{array},$$

$$E_{D,1} = \begin{array}{c} \\ Z_1 \\ Z_2 \\ Z_3 \end{array} \begin{array}{c} \begin{array}{ccc} \text{Rise} & \text{Decrease} & \text{Strict Maintenance} \end{array} \\ \begin{bmatrix} 0.9333173 & 0.06651228 & 0.0001704367 \\ 0.2858130 & 0.71365954 & 0.0005275002 \\ 0.5148470 & 0.48448983 & 0.0006631487 \end{bmatrix} \end{array},$$

$$T_{W,1} = \begin{array}{c} \\ Z_1 \\ Z_2 \\ Z_3 \end{array} \begin{array}{c} \begin{array}{ccc} Z_1 & Z_2 & Z_3 \end{array} \\ \begin{bmatrix} 0.01163584 & 0.6737522 & 0.3146119 \\ 0.33771812 & 0.4822758 & 0.1800061 \\ 0.11921039 & 0.1494697 & 0.7313199 \end{bmatrix} \end{array},$$

$$E_{W,1} = \begin{array}{c} \\ Z_1 \\ Z_2 \\ Z_3 \end{array} \begin{array}{c} \begin{array}{ccc} \text{Rise} & \text{Decrease} & \text{Strict Maintenance} \end{array} \\ \begin{bmatrix} 0.9522560 & 0.04774396 & 0 \\ 0.1675475 & 0.83245251 & 0 \\ 0.6474784 & 0.35252161 & 0 \end{bmatrix} \end{array},$$

$$T_{D,2} = \begin{array}{c} \\ \text{Strong Buy} \\ \text{Hold} \\ \text{Sell} \\ \text{Strong Sell} \end{array} \begin{array}{c} \begin{array}{cccc} \text{Strong Buy} & \text{Hold} & \text{Sell} & \text{Strong Sell} \end{array} \\ \begin{bmatrix} 0.3290894 & 0.09408954 & 0.30335348 & 0.27346754 \\ 0.1769126 & 0.49889887 & 0.20788678 & 0.11630174 \\ 0.3016659 & 0.23435906 & 0.40660242 & 0.05737264 \\ 0.6945444 & 0.15147877 & 0.05473665 & 0.09924022 \end{bmatrix} \end{array},$$

$$E_{D,2} = \begin{array}{c} \\ \text{Strong Buy} \\ \text{Hold} \\ \text{Sell} \\ \text{Strong Sell} \end{array} \begin{array}{c} \begin{array}{ccc} \text{Rise} & \text{Decrease} & \text{Strict Maintenance} \end{array} \\ \begin{bmatrix} 0.7693822 & 0.2302596 & 3.581568e-04 \\ 0.5733093 & 0.4265616 & 1.291195e-04 \\ 0.3968491 & 0.6018816 & 1.269266e-03 \\ 0.1018850 & 0.8981099 & 5.118758e-06 \end{bmatrix} \end{array},$$

$$T_{W,2} = \begin{array}{c} \\ \text{Strong Buy} \\ \text{Hold} \\ \text{Sell} \\ \text{Strong Sell} \end{array} \begin{array}{c} \begin{array}{cccc} \text{Strong Buy} & \text{Hold} & \text{Sell} & \text{Strong Sell} \end{array} \\ \begin{bmatrix} 0.4752889 & 0.1160113 & 0.2810055 & 0.12769431 \\ 0.1714853 & 0.5284784 & 0.2187428 & 0.08129354 \\ 0.2354395 & 0.2322489 & 0.4777731 & 0.05453847 \\ 0.3685638 & 0.1628444 & 0.1191914 & 0.34940038 \end{bmatrix} \end{array},$$

$$E_{W,2} = \begin{array}{c} \\ \text{Strong Buy} \\ \text{Hold} \\ \text{Sell} \\ \text{Strong Sell} \end{array} \begin{array}{c} \begin{array}{ccc} \text{Rise} & \text{Decrease} & \text{Strict Maintenance} \end{array} \\ \begin{bmatrix} 0.6309107 & 0.3690893 & 0 \\ 0.5527548 & 0.4472452 & 0 \\ 0.4567159 & 0.5432841 & 0 \\ 0.3846103 & 0.6153897 & 0 \end{bmatrix} \end{array}.$$

Chapter 7

Conclusions and future work

The main goal of this thesis was to examine the use of discrete time HMMs in stock trading algorithms. Furthermore, we also wanted to examine the use of hidden states as stock trading signals as opposed to the model in [2].

We defined a discrete time Markov model. We also gave a briefly description of a continuous time Markov model, before introducing a discrete time HMM with discrete observations.
We explained the mathematical properties two famous techniques used to solve problems related to discrete time HMMs with discrete observations, namely the Viterbi algorithm and the Baum-Welch algorithm. For the Viterbi algorithm, a simple example was given to illustrate the algorithm in practice. We also attempted a small scale study, where we saw the estimation accuracy for a few different HMMs. The true hidden state process had to be known to estimate the accuracy, thus we generated the process using R. In Appendix A.1, how to do this is explained. Two forecasting algorithms based on the Viterbi calculations were give as well. We also provided a short introduction to stock market data.

These topics was then implemented together in a stock trading algorithm. This was done by transforming continuous data, namely the historic closing prizes to

the S&P500 Index, into the discrete categories "Rise", "Decrease" and "Strict Maintenance". We then created two separate stock trading algorithms. Both used the Baum-Welch algorithm to obtain estimates of HMMs and the Viterbi algorithm to estimate the hidden state given the observation input. Both algorithms was measured comparing them to an third algorithm following the passive approach, namely the buy-and-hold approach.

The first algorithm, Algorithm I, was inspired by [2] and was supposed to be a close replica. We used the same number of states and the same criteria for producing signals which was then interpreted by the trading algorithm. We used the Viterbi calculations in conjunction with the estimated HMM to forecast the next day closing prize direction.

The second algorithm, Algorithm II, was developed by interpreting the hidden states as stock market signals, rather making predetermined rules of action based on the close prize direction forecast. Thus, a four state HMM with three observations was created, using the predicted state at the next time point as the determining factor for stock market behaviour. This was predicted using the Viterbi calculations and the estimated state transition matrices.

Examining the empirical results, it is evident that both trading algorithms managed to outperform the buy-and-hold strategy. This was measured comparing their rate-of-return. Algorithm I, though outperforming the buy-and-hold strategy, did not achieve a ROR close to the one achieved in [2]. Algorithm II outperformed both Algorithm I and the buy-and-hold strategy by a substantial margin.

The main conclusion of this thesis, made by analyzing the results, is that defining and interpreting the hidden states as signals for stock market adjustments turned out to be a good choice for this particular "stock". Forecasting states rather than forecasting observations gave an increased frequency of taking no action, which seems to have been beneficial. However, we can not entirely dismiss the approach of making stock market adjustments based on the predicted closing prize direction for the next day. After all, this approach had a larger ROR than the passive approach. Furthermore, this approach gave impressive

results in [2]. This suggests that we might not obtained the optimal solution for the HMMs in Algorithm I.

7.0.1 Future work

On the basis of the results in this thesis, we will now give some recommendations for future work.

- The main conclusion in this thesis is that defining the hidden states as signal for stock market behaviour had merit when applied to the S&P500 Index close prizes. Try to apply this algorithm to an actual stock rather than a index for further knowledge of the potential utilization for this approach.
- Apply these two approaches other data from the stock market. It can be another index or a stock. Examine which approach provides the better rate of return in that instance.
- Try to apply the approach of Algorithm II in Chapter 6, using a different state space. In Table 6.2, we can see that the "Strong Sell" state was predicted zero times during my experiment. This could suggest that the state space used is sub optimal, which should be investigated.

References

[1] José Pedro Alves. *Forex Market Prediction Using Multi Discrete Hidden Markov Models*. 2015.

[2] Luis Andrade. *Stock Market Index Trading Algorithm Using Discrete Hidden Markov Models and Technical Analysis*. 2017.

[3] *https://www.spglobal.com/spdji/en/indices/equity/sp-500/overview*. Download time: 06.01.2021, 16:29.

[4] Mark A. Pinsky, Samuel Karlin. *An introduction to Stochastic Modeling, 4th edition*. 2011.

[5] Andrew J. Viterbi. *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm*. 1967.

[6] Kevin Rudd. *The global financial crisis* [online]. The Monthly. Feb 2009: 20-29. Availability: <https://search.informit.com.au/documentSummary;dn=610838417780601;res=IELAPA> ISSN: 1832-3421. [cited 12 Jan 21]..

[7] Jason Fernando. *Relative Strength Index (RSI)*. Udpated Nov 17, 2020. Availability: <https://www.investopedia.com/terms/r/rsi.asp> [cited 19 Jan 21].

[8] Robby Sneiderman. *https://www.towardsdatascience.com/a-quick-introduction-to-time-series-analysis-d86e4ff5fdd*. Download time: 19.01.2021, 17:39.

[9] Australian Bureau of Statistics. *https://www.abs.gov.au/websitedbs/d3310114.nsf/home/time+series+analysis:+the+basics*. Download time: 20.01.2021, 13:43.

[10] Peter J. Brockwell, Richard A. Davis. *Introduction to Time Series and Forecasting, Third Edition*. 2016.

[11] Tzveta Iordanova. *https://www.investopedia.com/articles/trading/07/stationary.asp*. Download time: 20.01.2021, 16:14.

[12] Adam Hayes. *https://www.investopedia.com/terms/s/stock.asp*. Download time: 21.01.2021, 10:57.

[13] *https://www.upcounsel.com/why-do-corporations-sell-stock*. Download time: 21.01.2021, 12:03.

[14] David R. Harper. *https://www.investopedia.com/articles/basics/04/100804.asp*. Download time: 21.01.2021, 14:17.

[15] James Chen. *https://www.investopedia.com/terms/i/index.asp*. Download time: 21.01.2021, 15:11.

[16] Aaron Levitt. *https://www.investopedia.com/articles/investing/040313/how-adjust-your-portfolio-bear-or-bull-market.asp*. Download time: 21.01.2021, 17:21.

[17] Barclay Palmer. *https://www.investopedia.com/articles/03/072303.asp*. Download time: 21.01.2021, 17:39.

[18] Dr. Lin Himmelmann. *Package "HMM"*. CRAN-package. 2015.

[19] *https://finance.yahoo.com/quote/%5EGSPC/history/*.

[20] Olivier Cappé, Eric Moulines, Tobias Ryden. *Inference in Hidden Markov Models*. 2005.

Appendix A

Working with HMMs in R

R has been the preferred programming language in this thesis. R provides us with a very comprehensive library of statistical packages that makes programming very efficient and I have benefited greatly from a lot of these packages such the "hmm" package. Even with all these tools available, I still needed to create a lot of the programming from scratch. This Appendix provides you with some programming techniques I used throughout the thesis. I have limited this chapter to what I think would be of most common usage and left out the part which is very specific to this thesis only.

A.1 Generate a HMM problem

First we have to create the state transition matrix and the emission matrix, as well as calculating the stationary distribution. This is done very straightforwardly, however to understand the notation further on I have included this part as well.

Now we need to make a random state sequence. We draw a random uniform variable on a interval from 0 to 1. The value we obtain is an indicator that tells us what the next state should be. For the first observation we compare this

```r
#State-process
X <- matrix(nrow = 3, ncol = 3)
X[1,] <- c(0.5, 0.3, 0.2)
X[2,] <- c(0.3, 0.6, 0.1)
X[3,] <- c(0.2, 0.2, 0.6)
pi <- X%*%X
for (i in 1:100){
  pi <- pi%*%X
}

#----
#Observation-process
Y <- matrix(nrow = 3, ncol = 3)
Y[1,] <- c(0.7, 0.1, 0.2) #obs given state 1
Y[2,] <- c(0.2, 0.6, 0.2) #obs given state 2
Y[3,] <- c(0.1, 0.1, 0.8) #obs given state 3
```

Figure A.1: Creating the matrices. We calculate the stationary distribution, where "pi" is the placeholder

value to the stationary distribution. If the random uniform variable is lesser or equal to the fraction of time spent in the first state, we assign the hidden state at $t = 1$ to the first state. If not, we have check if the random uniform variable is lesser or equal to the fraction of time spent *in the first two states*. We can do this because we know that the variable is larger than the fraction of time spent in the first state, otherwise we would have already assign the hidden state to the first state. We continue with this logic until we have assign the first state for the hidden state sequence. The later states are determined using the transition matrix instead of the stationary distribution. Code illustration in Figure A.2.

We have generated a state sequence given our HMM. Next up we have to create a observation sequence in conjunction with our state sequence. This is done i a similar fashion, using a random uniform variable and the emission matrix. Code Illustration in Figure A.3.

We have now successfully generated a HMM problem. In this particular illustration of the program we used a $3x3$ state transition matrix and a $3x3$ emission matrix, however you can use any dimensions provided you tweak the for-loop accordingly.

```r
#State sequence
for (i in 1:n){
  if (i == 1){
    x[i] <- runif(1)
    if (x[i] <= pi[1,1]){
      x[i] <- 1
    }
    else if (x[i] <= pi[1,1] + pi[2,2]){
      x[i] <- 2
    }
    else{
      x[i] <- 3
    }
  }
  else{
    x[i] <- runif(1)
    if (x[i] <= X[x[i-1],1]){
      x[i] <- 1
    }
    else if (x[i] <= X[x[i-1],1] + X[x[i-1],2]){
      x[i] <- 2
    }
    else{
      x[i] <- 3
    }
  }
}
```

Figure A.2: Creating state sequence. Note that small x is the vector that contains the state sequence and needs to be initialized as a numeric of length = n before this for-loop. You need to determine n (length of hidden state sequence) beforehand as well.

```r
#Observation sequence
for (i in 1:n){
  y[i] <- runif(1)
  if (y[i] <= Y[x[i],1]){
    y[i] <- "A"
  }
  else if (y[i] <= Y[x[i],1] + Y[x[i],2]){
    y[i] <- "B"
  }
  else{
    y[i] <- "C"
  }
}
```

Figure A.3: Creating an observation sequence. Note that small is the observation sequence, which similarly to small x needs to be initialized before starting this loop

A.2 Forecast the next state using Viterbi and HMMs

The 'HMM' package in R is great tool. However, when using the 'viterbi()' function, only a string of estimated hidden states are returned. Including the Viterbi calculations in our forecasts, we had to develop a function from scratch to solve this issue. The code used in the function will be showcased below.

Standard Viterbi calculations

```
vitpred <- function(em, tr, sbls, sts, sprb, o){
  hmm <- initHMM(States = sts, Symbols = sbls, startProbs = sprb, emissionProbs = em, transProbs = tr)
  n <- length(o)
  oN <- length(sbls)
  s <- length(hmm$States)
  v <- matrix(nrow = s, ncol = n)
  symb <- numeric(n)
  for (i in 1:n){
    if (o[i] == "Rise"){
      symb[i] <- 1
    }
    else if (o[i] == "Decrease"){
      symb[i] <- 2
    }
    else {
      symb[i] <- 3
    }
  }
  v[1,1] <- log(hmm$emissionProbs[1,symb[1]]*hmm$startProbs[1])
  v[2,1] <- log(hmm$emissionProbs[2,symb[1]]*hmm$startProbs[2])
  v[3,1] <- log(hmm$emissionProbs[3,symb[1]]*hmm$startProbs[3])
```

Figure A.4: Keep in mind, this is defined for a three state HMM

```
btrck <- matrix(nrow = s, ncol = n-1)
bmax <- numeric(s)
vpath <- numeric(n)

#calculation step
for (i in 2:n){

  for (k in 1:s){
    bmax[k] <- -9999999
    for (z in 1:s){
      if (bmax[k] < (log(hmm$transProbs[z,k])+v[z,i-1])){
        bmax[k] <- log(hmm$transProbs[z,k])+v[z,i-1]
        btrck[z,i-1] <- z
      }
      else{}
    }
  }
  for (j in 1:s){
    v[j,i] <- log(hmm$emissionProbs[j,symb[i]])+bmax[j]
  }
}
```

Figure A.5

```
#obtain path step

pathend <- c(-99999999,0)
for (j in 1:s){
  if(pathend[1] < v[j,n]){
    pathend <- c(v[j,n],j)
  }
  else{}
}
for (j in 1:s){
  if(pathend[2] == j){
    vpath[n] <- hmm$States[j]
  }
  else{}
}

i <- n-1
pathstep <- pathend[2]
while(i>0.5){
  pathstep <- btrck[pathstep,i]
  vpath[i] <- hmm$States[pathstep]
  i <- i - 1
}
```

Figure A.6

```
#forecast
#finding best precedor state
fcaststst <- numeric(s); etran <- 0
for (i in 1:s){
  fcaststst[i] <- -9999999999
  for (j in 1:s){
    etran <- v[j,n] + log(tr[j,i])
    if (etran > fcaststst[i]){
      fcaststst[i] <- etran
    }
    else{
    }
  }
}
predstate <- c(-99999999,0)
for (i in 1:s){
  if (predstate[1]<fcaststst[i]){
    predstate <- c(fcaststst[i],i)
  }
}

#prediksjon
sts[predstate[2]]
```

Figure A.7: Forecast next state

Forecasting using viterbi calculations

```
#forecast
#finding best precedor state
fcaststst <- numeric(s); etran <- 0
for (i in 1:s){
  fcaststst[i] <- -9999999999
  for (j in 1:s){
    etran <- v[j,n] + log(tr[j,i])
    if (etran > fcaststst[i]){
      fcaststst[i] <- etran
    }
    else{
    }
  }
}
#finding most likely obs
fcastob <- matrix(nrow = s, ncol = 2); eobs <- 0
predobs <- c(-99999999,0)
for (i in 1:s){
  fcastob[i,1] <- -999999999
  for (j in 1:s){
    eobs <- fcaststst[i] + log(em[i,j])
    if (eobs > fcastob[i,1]){
      fcastob[i,1] <- eobs
      fcastob[i,2] <- j
      if (eobs > predobs[1]){
        predobs <- c(eobs,j)
      }
      else{}
    }
    else{}
  }
}

#prediksjon
sbls[predobs[2]]
```

Figure A.8: Forecast next observation